

# The SGRUD Thesis

SGRUD is Growing Rapidly Until Distinction.

Philip Schildkamp

## **Abstract**

Abstract.

# Contents

<b>Authorship</b>	<b>5</b>
Signature . . . . .	6
Public Key . . . . .	6
License . . . . .	8
<b>Preformatted</b>	<b>11</b>
<b>Table</b>	<b>12</b>
<b>List of Tables</b>	<b>13</b>
<b>List of Figures</b>	<b>14</b>
<b>List of References</b>	<b>15</b>
<b>Appendix</b>	<b>16</b>
@sgrud/bin Module . . . . .	16
construct . . . . .	16
kickstart . . . . .	17
postbuild . . . . .	18
runtimify . . . . .	19
universal . . . . .	20
@sgrud/bus Module . . . . .	21
Bus . . . . .	21
Bus . . . . .	25
BusHandler . . . . .	26
BusQuerier . . . . .	29
BusWorker . . . . .	31
Observe . . . . .	34
Publish . . . . .	35
Stream . . . . .	36
@sgrud/core Module . . . . .	37
Alias . . . . .	37
Assign . . . . .	38
Factor . . . . .	38
Http . . . . .	39
Http . . . . .	44
Kernel . . . . .	45

Kernel . . . . .	50
Linker . . . . .	52
Merge . . . . .	55
Mutable . . . . .	55
Provide . . . . .	55
Provide . . . . .	55
Provider . . . . .	56
Provider . . . . .	57
Proxy . . . . .	58
Registration . . . . .	59
Registry . . . . .	59
Singleton . . . . .	62
Spawn . . . . .	62
Symbol . . . . .	63
Target . . . . .	63
Target . . . . .	64
Thread . . . . .	64
Thread . . . . .	65
Transit . . . . .	65
TypeOf . . . . .	67
assign . . . . .	74
pluralize . . . . .	75
provide . . . . .	75
semver . . . . .	75
@sgrud/data Module . . . . .	76
Enum . . . . .	76
HasMany . . . . .	77
HasOne . . . . .	77
HttpQuerier . . . . .	78
Model . . . . .	81
Model . . . . .	97
Property . . . . .	110
Property . . . . .	111
Querier . . . . .	111
Querier . . . . .	113
enumerate . . . . .	114
hasMany . . . . .	115
hasOne . . . . .	115
property . . . . .	115
@sgrud/shell Module . . . . .	115
Attribute . . . . .	116
Catch . . . . .	116
Catch . . . . .	117
CatchQueue . . . . .	117
Component . . . . .	119
Component . . . . .	120
CustomElementTagName . . . . .	123
Fluctuate . . . . .	123

HTMLElementTagName . . . . .	124
JSX . . . . .	124
Queue . . . . .	125
Reference . . . . .	126
Resolve . . . . .	127
Resolve . . . . .	127
ResolveQueue . . . . .	128
Route . . . . .	130
Route . . . . .	130
Router . . . . .	132
Router . . . . .	138
RouterLink . . . . .	142
RouterOutlet . . . . .	143
component . . . . .	144
createElement . . . . .	144
createFragment . . . . .	145
customElements . . . . .	145
html . . . . .	146
references . . . . .	146
render . . . . .	146
route . . . . .	147
@sgrud/state Module . . . . .	147
DispatchEffect . . . . .	147
Effect . . . . .	148
FetchEffect . . . . .	150
Implant . . . . .	150
IndexedDB . . . . .	151
SQLite3 . . . . .	154
StateEffect . . . . .	157
StateHandler . . . . .	158
StateWorker . . . . .	162
Stateful . . . . .	168
Store . . . . .	169
Store . . . . .	171
Test Coverage . . . . .	174

<b>Index</b>	<b>176</b>
--------------	------------

# Authorship

# Signature

-----BEGIN PGP SIGNED MESSAGE-----

I hereby declare in lieu of an oath that I have written this thesis independently and have not used any sources or aids other than those specified. I have always marked as borrowed, indicating the source in each case, passages of my thesis which are taken in terms of wording or meaning from any other works and sources, including sources from the Internet. The same applies analogously to tables, maps and figures. I have not submitted this thesis in the same or a similar form or in excerpts as part of another examination.

/s/ Philip Schildkamp <philip.schildkamp@uni-koeln.de>

-----BEGIN PGP SIGNATURE-----

iQIzBAEBAGAdFiEE0GFzy5VlTUj/jnYCdWVwy9g0gMUFAMSMIIkACgKQdWVwy9g0gMWDhRAAGAtjeKRX0D+e5U/7LdPVogCTbHVqUPSuhIXfWZgRMqGibuouB7WaMIRiWk5qQ0wSQChTevGPgCkmpOMFxnJcp+YBwzrx2bsw/0H692pIeQXKXif4V05QS8UNjXrfSw7pBTj0f5t8DtQwxSlwKMmt0VMUKRhrEz3IuhoAXuxFpw+F54/St8AhGAfoYHl3/Qca4IQWzuG5wHkMqMe2mnAACLDD/cTpKbj0Uv01k8NkzR62nLkoLCiC3yNpQ/rMY81VTPJq0j6ZqadAkUeOfJZC09D/pvFDBYEclGbrVanuFoKP7DwPH1j+7ljCvDwyTUUUcmeE0Stcag/LITK8xBE2YmZs30FREKXNnnUp3RBQ+Na/t/DwYuCLRPYyVa9cSvh06dCT1dYmVQHln/ZvW3j/RpPpENjxAyozZelx7kNw6H73ocDSOqNI+gePD+C/QhNLJD1C0JZ6f1o9jEeF0/P2z08sQSPmAFKs/vaxy9EpmPleQILNUbePv7fsl08cK8TkuHQIgE1x5miAhjwKpcemOuWQLE0WCgt6Y3s8jxKIakW2izZ1jKXeTArNaZuY4PFMW+KQgmZC2htHSLxn3QLLsDEbxeU13S8UC8K5Z0b7+VK25fyUKZ3m0n3mVyoSbko2/2t8pBvr/mkMZEc4/jB0dpHwgMd8TXKqxnSkvZ3Tvs==aASQ

-----END PGP SIGNATURE-----

# Public Key

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFI0nqMBEACRTgEo1MAkiyLVfyxIryMeFipYfd+yZdSLIYls8x3DAOLrK25EyRmNL/qzL5zucB8WY4j+cGQgzT6an/+00rhX0lCRxMVVKUCPSFCDCAEA8/E1k3/uaHQChNMQodiVx9Drm76kLA+sxhnZh0Ku2C114p6hofhstTdVEfKOYDxw1Ufzky5U+d9+ilKBS+/aUGI85XzLyYMWtVdDuheQ5d8A05y6xGeBStzy7cF8D+44ojdWkrJplwqild6wJo6jLoWnuA8K8Fw6Pg7CxB1AzledHloLaL6yG6B41/bSbQmxGsxpYBD/U0S6nQ+/tYrXlGPMrAra+PbNy0WYdw+JwVB18jSluQ8QxppgPqf5fJfLMQIqARN7VIX8z1EyxNGSsYH5/8icwuqoVyFzfKkjnTaAVMDrPA0oI8JZq2HI3izQif0STG4UVEw+ixgZeh7kRKANJ+Xu4rUwAxe7QAGNBKjB6yMRZLS+rZ3IIdxGfAh6piEaofB+dx03C8KwarrBQSuLE15yINGMcA/r6is0HYhrfBu73whdF2d6RamXmdCQYwIta7+FxIHbvm8sNF/yym5DxvLSNiLT939ecmBG/zcJHEAi8r5Bph57CSFsCpsphW6dFwMt2RG415KYTH2jds8a3jfd1HHAuAzT4YIt5Lwhr4/EYzyiJQJujSjvQARAQABTjCpQaGlsaXAgU2NoawXka2FtcCAoUHQpdmF0ZSkpGvVnb0BwaGlzLmLzdD6JAlcEEwIAIEAECGy8FCRdGvLAFcwkIBWIGFQJOCAsCBByCAwECHGECFAWIQQRfi0fi0rhzmFF7aU8tut90hMSgAUCX7ubFwIZAQAQCRA8tut90hMSgAlgd/0XGZ0/LJjppnoyce+K0DB4r5DBmp2axqGeDZUpLGRF8ZNL3sWn3MbVteBpdiTwWB+97+gpYVNOt6gw7+7BgMpnAcypF3PKRzmQMwPAZ8unxdw4GEe8+2pSEK790ifwEsXPICG2WF1YnAE+P52+SVAgQMT0uLXKwcBabUS2+yT85CNutU5bZTwrnQJERRJnzdz06HwMMULHc0aM50RV5RnbknXK4dDvJgdVD3zbnt0adiqK5gw/WpFM3ir/GzXb6gEtAkZJ3DL74ICP0qjde9WmX3g49iRpVmpvjlyGxS+ZggqFZUMndRjbuwQI7acJICgQzbe4SnAX9sUxjJdJF1B8hYT7Q6LZ5wg6MIK0Tv+ABxrmCoE9VgcmRir2We/QHwV7AkXqYmNTGIL29AGCH4H+Dqx2D/2Tx/8/WxBL1SoralaaPRzuwGB6CDVooUSyGrHfShk85KA0Gd0IutsaufeKPK9pITLr8k8i00uD17J4o+6nxwUWZy0uxCMF0jAwzLXCkAuxYkgoX0Rq7PVozT1DwShXk/KyyWSTyguIvvhEhQlfjJGOWpInJZuyh1hEw6C10jwPnF1qSDR20Ev1cPaayep93UgJ/rleFByBzeRh/zkh1qijoiLFnSBiWztpKh+flhtrhiQ6HAepwKMS0PCGzUP64ijhozbwcd1IyK8jqokCMwQQAoAHRYhBBoF/F7NPDv08VMPNoT+06L8YD7fBQJgECwAAoJEIT+06L8YD7fCz0QANAFY02BiQXsrZm0IdvHndaS7p0CT3EmBH0y4m8h+Ey8L3Ueejx0vbP68ZhxEokq3orD+7erRcD1VngpnGszcwiYym5ZBSP4kIka906/dtKkeFISgIkYP/jWof0hvs90tEW5kfzVf1exmeM30rFcZ7kzgx4psvQqUChJdxn2Iw1Znsh8D4enARU5i8A3b0LMPtk1phmwq8nVJV5P1I0qWSHetcp4LZQkw2hanoeYCPyfTqWjusaQvfvGzksOL1Kb2cGfI3Aji1yRSKS7vB4I3ld0tQb3xFUXQBhwVqogZwsVa6+WoUR63EozVOLv3ly9ubn2QAudv2l8vGtDf0v0pEPRK+QZf4DqYKbhm7S6A0sqrJlYbhoiRuTFxxIavszQhy9ob1oiqLphQo+KQfkc6+PjB83DV0tjJRMnmgNSHRT9nyHJR0e+pwbm671B2A5yZR3K0jqdD2jT6zYemMI4M/U3r5MIHAq+h1SRA90x6XfFAYNDjJaPVMPglQbJkr/XOFB0G1F/F/xfcAuqICE5dNs18gQnou4WkpE0Lvm0y951JctAd5gFk6v6B3JzeElm9igOn+AfYyZ+aQImQ1GBDe0Q/QMAN53b+gx/mvQ07mFnXhD4phEQU94IPB51zHCVegnFHDIZJLmuM6n89XQ4m7jvC1S0LiQiCBBMBCAAGBQJGec7jAAoJEF5CzLSkv0PX2uQP/0cpjsbZrjURU1Eny2/v3drDaVrh4wKwL+1qSLhp+4sIKESmN8FSCfMz5GBQv+CSMKDusXaUInM+HiRqVLcuuVOM3E531jQE5+w/7wj0Zxz0mDJUyCshQ3p4puRtw7FU9sL9073JVxp/4EzqYHl1p9CMCjLlI0it0/TZ5J43rLONNMZSF/9ui+ZaRU+CAHTUzsb2DIIti5kLaGpQNYHbWQjsNga5XxzQ7LD3MvEsucwzPIJv6bgUV5wZzqFEYQH01JYRcTf/6KgQr4080lZohAirSyCbNrgbsTrK0UMFjen3E+1FByBfrjRwXZzHbF0xgjLcaAYjZex02Pz9SKVIjE6mdo1f+ZoCKudCNvLMxsCL7PCUricqfDVRh5ECq4E0nezRBK2sw0Z2NEATe6E5BoiGFvoE1QI7oxz6Na1v8iy2NakInKdevFIJENZ+XxRmAj1dLRSf3v2sEnrFz8PtLj2n2Ih74pAmK8Hp+Df2izQvys8f0hSuZQfiwMxkvy51EUZwJZp80admJNMjmQHwgthrlQwPH4b0LWN4e7YswLnV0zcZjCwj3bLpJyhatHeTIWP9zhRvRvUaWeAwd94w6aQb//rpA7EmpILtxrKI+S5SS/P8LLeFpL5wEC94K9baU8Jb1zx1UNCbeL84QIH2Gnmr4x8FvHgvZ9TPzLziQIZBBABCAAdFiEEqEnuIodwANJnVdsANiW8d0tX4YIFAMAhYfWACgKQNIW8d0tX4YJoYA//SLAWYUHF1dkwW1ImXD+tgE7YNJo9hWYdeL/pum1bDUDTEtq0ZVe47N2aL50B133foL/kPa31IGaHPQEe/WJyHTTDqZ5qmmrDvsMm8AcMczVL49zkijPKyFnmmtCtHnSsFConCxjua+gznmJWPPfHcKTI+xX1UD9uj2iYBHvnao2Cy7nTMaz6J2U78eTvB3cwLWU6pucJa/AkI43ZDYm1Vot1teGHgYfCsJxag+nfv4UI56vm0kquDeqGuyc9TG0CW72cmVGJ0inJ3pwtNL0ger83chy/CiRILPGogRrtF4vKJYxxp38S2Va/KXpe5UnatILuV30su+aNsSsv03RPGEtqi8h5ezGqXW8p7VSizl5zixv3SEI72zfMfoDyFp0/UIlFYavCq8SiRB001yhGRzTESxUuTCQZegFnyHsOd1mC8JkgHh9+2iInvcyt0XQdZvuBnFhC1mPrQsKmo0ebvLSF1PktYzobB7c4rPc68jJcKy5Dm2Za/DzB505+Z9FJC/h4keKJdskRI6S9L208d2S0tWdy6bZIJ4dXSshdn8ILIUaZ3zW3wzVewjC4k3ehyqI0W7TcrBpy20vcVDwygJtSpmmnAnL9T01BP6zX4uKmdKQbuEpXFQziNGDDxkKbKne6jXkU8RzNfMaeQ+2CK2yq4tddzgHgAcVxGeS00VBoawxpcCBTY2hpbGRYrYwIChDb2Rlc2NoobHVzcYbHYlIPIDxwaGlsQGnvZGVzY2hsdXNzLmRlP0kCbQQTaQgAVwIbLwUJF0a+UAULCQgHAgYVCgkCwIEFgIDAQIEaQIXgBgYagtwcZovL2tlexMub3BlbnBncC5vcmcwIQQRfi0fi0rhzmFF7aU8tut90hMSgAUCX7ubFwAKCRA8tut90hMSgClLD/9p7Au7c8J9sbdnuYn086XMDcE9MMPK5+I0S5y7ZBAPYR76nh95M2gveRBY6LmRTXNaRPLxpGQad+9rkiQSBY0qm2MkLxcFL+Zd8PFnlLHGEG1Y0cpeXtk/Bt+RPhao9LHrcNn32V83CoqZ/OWEErIt7vtrpNy+LJt2UduFuji0Yk0fNX5fJspL4A63PRdnrTNGxstMsNQPjnLCZ6FJRLbgo8/Xg6QVUF5IRK/adATzAwFc3MvXhZyq/21z1S2+9WUZ5ow30TC8TIgm5t7Z60y8DervmIFx4nYpOuTLi6NpakCNKuDE841sFz2aL2FPWxNurLI8UvW7bcwJ1WvwxET9S0w0XYL1KhDBLnpZpDfGy2xpYynbgSePCAkuY0FjcBfbM0848GrA3E9yXhkdRiYkPNsbgZaQ1N1INiriEni+h6hkPPFVUo9ghEYAMBf4hiHE/jm83qCsAPdp13Td0780sikQbVnNpfiCYkFA8ChM1Cp7vft/eUUGF3zKPTsfGYTqWuQvqZQ04YnLI9QjGdF6GaZ4J15fsqt0sr

70/fFVYVxbuno3sLaIpczbnzgcLAbIKx0pt5S3wXl0YM0RwxvLDHBJTFZ51uukoIHK915G1fJ+foQrJuK0zLHF/jzQctWC4UDD3MaDhNrZVCg  
LZ/yrDp4h9a1/wHuYy0m05YkCmWQQAQoAHRYhBBOf/F7NPdV08VMPNoT+O6L8YD7fBQJgECwAAoJEIT+O6L8YD7fjrKQAKcuIBLVQYWBRIzF  
xamvbcC15r4Lni1TA03D0SjgCCqodwhe1JKkLUUEM83orgdPZvPo65EQwHtB/FnN0xtP+z2QH3teLPj9/6SXPeOFNQDNYIooP02wJ2EQTUDX  
UE/heSvY5Mt2CM216ZbL1TkU47TTfMBSxYsVxZk39vLiNdN5HW80BoAoIX+EKuGXUXPe16Z5ivHmPis4ru05cWpnh0+ccHiCJGiMkwb6k6Mfv  
shw5kDYaCVDSeRjQv+ebiPLyeDmZeJ/KzG7+NKabHNpCabTiSdmJdH6B38C9+r+/60RC1NK/JcPz33WobITyn/JF05MZDpkjN0cMBZ6n1g6kW  
JRrD2FnU5jqlU6+IntRWffgJufMCTsmLgpEvFaNqAcw9eHQFMr8w3s3ruacq04cK3ghr15SN1q/mFDLPQWiIrgB8D/cuPC6jw6bKQ8e0oAVG  
kQ8pltljqlw1ynq/ziwyFKUFs2vIQRty26BE30gbdJrWsaZEUjP7rFz2Rw/997DUH+2hgsz2aE/rjXJfTAqrk4WdhZLm4aLk/YLwYtByeRpXop  
97YpVDLXIvVZQeGM5Ka8A9xDMRBOuYb7+SPqsr609AZa00VgNVD+vK/rN0WBcfubUDLybh21Vu9pa+0WK0DvhX31uFD2Vj/htQ/ZgjZ5dYD  
3c5V8X/2PNOCzEiQICBBMBCAAGBQJgEdF6AAoJEF5cZLSkv0PX0igQAKFFnG7QLVU6NX9xm+Azj50XSwhbUtMy88xxu1lS0IfIzeSbY4kn5aA  
3j6xAs8hnIXtPLJsXDIKORiF0MzChTqL3zJSh4Es8sDme70aXAETGJXgeIZXmih1ZL2SkHaDviouTJx2xwxgrXv8LmTWuAy+HgBkuC1fAIt6  
VKZiHvvfE6JX3HYKJ2EAWo1/81DuNBXyCA9pyjZMXEIJKG60UQm+7PLoN+24M09DLZjwD2c7EAQ1ne26IJ/FDHLdA1086PI76bNu11mov  
zDtyT+KDBxogTUYjUI00pI8VFXLCmuB+XlBkMd5hPxDF0ykQ5mqtcXFNXSg5zuYagMa1a9rxgwtUShtvZLpi0obDnXKnggDno13L56SFbbWw  
pv09rfMG+2D7b+/j1KP5SMwgmPYBSvFqYhcaT5bNSribSjktjAyaon0A30Ag+RsLj+qJEQAv+ueW4Du8xc/eFoTdfKePJXwxzpkglqX1gh30+  
UKzr0lqrZTYeIggwB+YrDj8NyDJSagBC8ZiBlI40mFEy14t8WexkNLfrn8Uui4NrR0SmXfCkF6+ULBSeqjg0LW8fkHmhMSLxEnyiezFQeddhvb  
XMiEuVNa1kat1PEaIYPF6DvTBNpEupKIFTT9k5U+Afo/UKe84L9ZAQnF/hQhRr1+pZZJLbLmNd0gR91x3eh4iQIZBBABCAADFIEEGEnuiodwa  
NJnVdsanIw8d0tX4YIFAmAhYfWAcGkQNIW8d0tX4YJsbA//SyvPn02k11yn2T5+iNLo/eTQP91T39SWNIF4dyrpkP0I03TKTzqo5CZDeABRE  
LTX9KZ/7ZL7r89Lnf4fzWm7hNthgkGBiZW1LHG66pX+39JV4msKvcrM4i0OpMKGHESQqbcPq57f716AZ+5RodbUXuK92xohS6X80dBP8/Shg  
pCmWL62+KzwyTM0YuhNXlaFfbWsvV/Ny0nhcY66BALOPFBGnu2ouVhj0BBRBM7R7uKu9mdtb/jzLXIm2tVkeevNjtBjvyRoge2BRT0wG0mGy3  
RHNBQBF8ykeV0V3xoyk4K4A5mNnbF4xkzEACPhqT5wLB6/77hVg84RZS36ag1a+WBE6BZWH4g0jByRf/pemrYmJtW4V96qUrP/Q/KyZgoU/J  
10nHqQ6TjYJA6uWnKYLHqPLOYGdW05Ea0DVxdVc3jVUoHdcL5i1ZQVHT1xR08bqVxQi5zmTI6I+nxFPndZccLA6GKUuHhKckaltvzXoT/ZnV  
mPuwUHH38EL7L3tIMGsufQnrKazoue9KDx03gSMLi+J8d8zk8WcQ4NGN9hGXdG3DMP7K15T90L9npvdGCqKysutI6imV05UYaZ6Y/CuH2L3S0  
RC3C+JrwTMWF7kiJj9r+E20h2sZYROSLeM/RGJ7ttK8YlMBA9TLScw1xwIG0Ywgl34YstSVL1Ay0SLBoawxpcCBTY2hpbGRYw1wIChVbm12  
ZJzAXR5IG9mIENvbG9nbnmUPIDxwaGlsaxAuc2Noawxka2FtCEB1bmktaz9lB64UGZU+iQJtBBMBCABXahsvBQkXRr5QBQsJCACBhUKQGLA  
gQWAGMBh4ABAheAGBh0a3Bz0i8va2V5cy5vcGVucGdwlM9yZxYhBCT+LR+I6uH0Z8XtpTy2633SEXKABQJfuf5s0AAoJEDY2633SEXKABQJf/2  
wElCsjj+6j5j7LI3hoTvGSH4RFv925z0aBYjppqdLWpQ/iFwd8nbLvtH3Z63naEvdPsfS0JW6ec3xKDDmkIkeAUPKM8i06Pf6Sm5EZA64oDTM  
3r2utCLfNBu9D738X1Cs+86KBKrw8hQtMoWyKJ0dCdvi4XbHmxjBE7bjjs6Nk52zF6sYjqTYFRE0wQuXwDRR9uZft3y83aBoQZAPHqzrbop+9U  
k88zV6aBn3u7b1n+DK60FK4zhYRiLmt08NC95ljNV40i5FYbPGwef1a3l2S204tthtYAFr8YKx8f1fFeCz8rqiLkaldxvj17W1GLaAR5u+bJAM  
XcstK9pc/RNSwFEFTV2AdZbBSsfv2mkArUq9tP+bgdAt+eiKdKfDMZPZj4NgiprUbEvRS3UsffXCLn0qSkNvSM1wbGpGZI7ea6+au9qzW3mU  
KgwJ1B6qgA29++eqR/0ggUslJ9S6gVBKuTuyDJxKjIjF57sHL3wvZAB8hSzq4SrbivIUTvyuITChqMn0pZ7+PzYCrOUFFGRcXbypFH5/dJ6M7  
964pH4jo/ebtK7b1y3LK/FLWkxi9utfvQ/ng9gWU03ygrlM8BkFqTbi0D5MzTo9XQoQfMiwI9dZFKawz1XfnfYkRmVD0iioeX0/D3235hmcot  
WC9JlmeHnq9oYokU5p2XkatF/4JpiQIZBBABCgAdFIEEG6X8Xs091U7xUw82hP47ovxgPt8FAMAQLBYACgkQhP47ovxgPt8U+//cRZGA+cmp  
Ou9LVk6f6xAMY3+73pbPrbdc4BQ0XUYiusLPGQ22c0ZULei0YyBdu98aVKTo8QcJgle6LETdvbmibBL6ej4tyeuXvFzARCoMZvyrRnrHnYv  
AQQVKMh0rr60KjAhaEJFnFnRwZFTSS03UMXjjskWLmUoIPtULV0YaJdAyeuLmGwRPAxqIdWqEowtk/rSjXAlu+EFzWRP5PtK5m5vryDQZCymq  
eLwAx/Xszg3mfKJheLX390h/4YQh2i1k9KLlWlRscovo0tJjFnmKF+lvVgDX3a9rctafj09MUUjJLS05Pdcv/3Hd0hNTTbmsctpq7obhtdU8  
2kjT/DVmIaBrkWAztC6KZDFLk8ipC0F0ks7nglIj6D0aJnm0iDDrDXfDlFx+LJ62JzHDxn4uYcD3iruz772zuNh4kNHa7ms0CD6CU2odfma/  
HRXsl0/M8rIdyd0CpOS5/Uua9NX02qZT9TDiMpoDC2Xow6fRFYp5AQQBX1wck92DUJ32QstxvMGgtA8IPkPdW5gJ6ooGbzY7hTgIeBQtSBSN4  
znlX54rKsqX+1Elk/AK7Keb0krQdq8BpCmu0QBR0JtGvNvVncFCs6GwtGaFqL2DJGo7A09NAdLCyv/AMRmTWHRZdzkzm9Z/h0KEor1xDHBC  
VRWH/P6767F5FLKqPLLGJAHWEEWIAAYFAMARLYAACGkQXlZMtKS/Q9ccrrw/Zn0vSDzUMBeL3pXPjYzL34xPK+Z+3M65toFA5rXeF5PLLPke  
IdU07PRVK60wArYVI5FFLASnjQxxs5Pff90ew/rTsQiglsUAMZq/f6DVVfd4Vfexpthjt96JzjZ+gk3njcn8Ngl6ndz2kjs0yZbPG+KQ2kAO  
UN2ANRH3db4XLdPtq0G3/Rn0o1w4Uz8mNnLBAR8RSNAu0VYH6+prAEA+1/84tGrhs2da1WN1PRs+Lv+E7DDRajQPz2w/awK4ymYfDM0jmS  
S6Nu7Fd4/a09s1anBFRGDK0HlKpTm97abFXJi/yuj8o5JQcK8IRbZ/1ncD6558qo0K0xPwJL6737UFyU/48Ei10uHMZA13igsfkoYEMaBa/  
6nLibi2w50LId7CkxXQNvMhWR8hQX2GfsgU2wmnrw19Yd9anNXMVHBG9IetGwuAKZvhevVoZ6yC9HwCS1dwDaoIu0tu0/tKJajIhH6CnT1dnHE  
+HN+qPwElttcsNNrFA7tnYpAFblnTzI/C7WeiM6ari6XjDRM0ii+2L6L4qJqW49mwnjm95DTUn1JufSneAN6qsjsJwoIwF0XhUkNZ3AZaU5  
17SiByMJNI58gVs+wRAV5PmwFXM4owXutnZDF5CGeDBbzFawLRyGSVshPowioe1nagjzayq3qYX8J2fZx5IQ6Dc5VaJAJMEEAEIAB0WIQSoSe  
4ih3AA0mdU0x00hbX061fhggUCYCFh/AAKCRa0hbX061fhgqNtD/9RXwP+kDRG9FWX5mcH0XWCS21CW6H7k2CdXBa+6SBqX6byu9yIcJmS6g  
4AUfmwM3VmwjPmuIVXK7wSr2Gh9EM/d7/4Rh96dTMqMneNCXLtJbMSfAELTLLEfWbXurZDQRJ6mwvGLE90ZbFvWkbP/hF+04aw1Ru6fSkyA  
SZ18L0Fgt7Idr7uqJ2VgaP/nqxq0mAeyP3FdgC/3Vc1D8m2R1bZLmPlr+nSpi+GxmtGvvu/bSZEPM7j13vFjms7Rz7sjeUcUWJZDVnYg8WB  
cBXsPTNnEJmW0IJo7m65Yu/IvM19R0SphTese5Qvq00QCjpljMktYs7d4TD/keBnw72KTo8xuH15U43AwbW7748Y5hsJXIEY3fBb/hDIPsv  
jt0Q/c1ScjjrZJwR3U57DJXZPKMy+lKnWFS2dnraBk1FqqdF3zGHEWLe3C6q35E6XZA405qevBgpoRdkrufXCr3BF/0K5kE8HAPvRDC9qY  
fxGFFVi+twOYL38yuImZe3PMOmhdQsV2N3c1eTbaS6i13fFy7kbWbf7DDp7tj2a/btiEblXZ06eLZYcktyAwN+IqSD3f65KasHnPTxwFwW  
fWJRj0Gv6+1qS8v9Vj3s0JLReVU81fV0ymdt0L2XUeM3hIbn4MU3yqKXGZwLThwbNlJ2UNV5g5VCIGhbW47k6CDQRQSDp6jARAAT6K79DeIImDM0  
nc8HjzQ5LY9S4PgjoLVtXq23JXEPj1knSd6ACYtr4MuirhVLM0u069PNKMLJLPDDsMRMLXLB/GWg65K0Jmpu0BwB6I3JQWCKtXRdHmTbT  
k6FlxjPv0SB5ZpulG+P4tgVizPGTGSiZeh4STYUyHK/fwkN//h9SiFliaPfd2Z06/113L6YrVI7WBeL0TEyU1vt/gytmrst/gEfN1Yjb0HggC  
VUkckbmabMJE1xwcL+ZUWKGoE3ICv0WufWwIiOvDuniTOKUHthb0up2LXJqon0V1Fyjt6wc/kM20p0HU1w3D0EJIKWxr+1PJSS13cAZLW3Q1w  
D77HE4S2zVGZkBgG5dcJyTAW5Dx8pqYyvCceMdmHG9KIYSUjN/Vl+MkoyXmEcLj0rfaIhAneF13orKK95AvixfPrpVoZfAQeYqMdnJQV2fGwRT  
i5I2XACUzzW9EQFru9riuxRSL/MJF7Apm4KSTix7HE3wg54sFCBc2JAE3TVD0e0Xy9u0mqMnVe/toSe43wfJG6t0MnKnuucNEKzIkVNLWukG  
MdcTzoCROMSXQ23+WEGzP4PX86JkILRpev4oxr3SpExIpki76u1ppc+Rdwn8ZL85gtyoTOHvokm35FIpYYG1b1oadZqMYvCnrWr+6feoroM0  
rlnxd+ZJM4itIuMsAEQEAAYKEWwYQAQoAJgIbLhYhBCT+LR+I6uH0Z8XtpTy2633SEXKABQJdoLuFBQkXRr5iAinBXSAEGQEKAAYFAlI0nqMA  
CgkQdWwVY9g0mUCyhaAoZXyUwBytaoVc76yDptiY9saTj2dEmdN97wg9oer7ETf0Yc1MyZKJtm27+Ty4I3uuikBaQsIce550ZucMUE4YmXA  
f9AZyZeEknTAujTgfva0AijzW9p45NVnURCwRamlA50lbJ7PHZOGMHUzqlfr+TmbFl6DXKeXQLknkXuy3GQp2Rpy0A/Z8+VIimB7JEAeqPjWe  
M7237Uf1h7rHJ/lrMveX7HYUvDq+gwm0HWLGPopp6CkIhYL9of0EC12+AtfvD5027TdjiaAayD0+rSbt9b0MFjCNXihLKF+ecgzQFo0+ka  
8ikplgz/k82at5NCbnv5FWJGh5D6TonGGcryct1rcgz9r3eikc4ERCp2SgnfQ0UcLLe7V37LP/7vxtDXhB+ntdLkh+PdhIIZ6257/1u/JyBe  
xEHuxKvmM56h/DVK1/4J0cIEouYpF7FaEuQ18C8SdC0iC0nNop/Pqxzg4DBUIjVKSyCtRoty7cnzEKvTWLFWmoDoL5d7I87TX00u3gxIac4i  
n6dmisEPH+6g9NG6YAP3QHTYRT5Suzc0cT2f10l8aUngpfEmVcWQ31aCw95gdbTJfSKwa7L0VwjnYqhaqCBZ/jlbmaU/YBx2UU0LsbP80a+h  
JfSD+HBBP05YErN78fMvAkfjxhF+WGMSQuYBo3z7wm38+oeSgJEDY2633SEXKAXHAQIAIBIdwk2dksTud7g2fj+AQ2Bm94iDGFWpa5T9Y/or  
jJgI3C1k/x34at+SB5GJHLYlyPbHdThkqEr0GEaPxtfW5CTma+VdmJpTcgghYhndQ2MpOUWUGCBMU+IpAHRc8CLOPz7Y+/LC/qYrtykM3a1L  
7Noy9aSJcdYdeBdVf7pyqv3rvjpcjVk/KV0tbZBehxd6XtCLV+caPRsraoSb2y1J++e28RzDDgLIJ8ytSjNiz3EQXud6HCC46pp0Mhanix3d  
vdJ4anFmeGxJpHMU4tWABJWziE7Ryge7tcNrV0cyK0jCMC9fCvNP97JaE35vsnf6hDqEag34avdvtnl2qu49ZiQqbmtNgHuBguB6I+SdqJ0  
UysiW6X5XnQV1chAefABK/0iufZSMWohry8b0SZX6BI0587Gmo4LXIDAF1dZv1FXeMC9vm9kIYv9zxuxNM9gnd05L9uKZDH/4A65DF7HPpaS  
ksugjQwSABscEqn6yWAmLEpPqthW8qbxZKRIA6tVu1927hPTveyTfL8vmfcZLv8gmQdiRMVafmBoHJY80JUZwocwheLb9Nse2+KZwnf1laXD  
ZsY2Mjwbq/4Lx/acAK4AFkkTnjD0smwqPYsmKV8/pWoAsv5DXR4e41nEW9BBQZTpGve0IjPit0jKVnwR3ynRK10BbhccPhe=Z0Cw  
-----END PGP PUBLIC KEY BLOCK-----

# License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License (“Public License”). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

## Section 1 – Definitions.

- a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.
- b. Adapter’s License means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.
- c. BY-NC-SA Compatible License means a license listed at [creativecommons.org/compatiblelicenses](https://creativecommons.org/compatiblelicenses), approved by Creative Commons as essentially the equivalent of this Public License.
- d. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.
- e. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.
- f. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.
- g. License Elements means the license attributes listed in the name of a Creative Commons Public License. The License Elements of this Public License are Attribution, NonCommercial, and ShareAlike.
- h. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.
- i. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.
- j. Licensor means the individual(s) or entity(ies) granting rights under this Public License.
- k. NonCommercial means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange.
- l. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.
- m. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.
- n. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

## Section 2 – Scope.

- a. License grant.
  1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:
    - a. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and
    - b. produce, reproduce, and Share Adapted Material for NonCommercial purposes only.
  2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
  3. Term. The term of this Public License is specified in Section 6(a).
  4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to



exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

- a. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.
- b. Additional offer from the Licensor – Adapted Material. Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter’s License You apply.
- c. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
2. Patent and trademark rights are not licensed under this Public License.
3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties, including when the Licensed Material is used other than for NonCommercial purposes.

### Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:
  - a. retain the following if it is supplied by the Licensor with the Licensed Material:
    - i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);
    - ii. a copyright notice;
    - iii. a notice that refers to this Public License;
    - iv. a notice that refers to the disclaimer of warranties;
    - v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;
  - b. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
  - c. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.
2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.
3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

b. ShareAlike.

In addition to the conditions in Section 3(a), if You Share Adapted Material You produce, the following conditions also apply.

1. The Adapter’s License You apply must be a Creative Commons license with the same License Elements, this version or later, or a BY-NC-SA Compatible License.
2. You must include the text of, or the URI or hyperlink to, the Adapter’s License You apply. You may satisfy this condition in any reasonable manner based on the medium, means, and context in which You Share Adapted Material.
3. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, Adapted Material that restrict exercise of the rights granted under the Adapter’s License You apply.

**Section 4 – Sui Generis Database Rights.**

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

- a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database for NonCommercial purposes only;
- b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material, including for purposes of Section 3(b); and
- c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

**Section 5 – Disclaimer of Warranties and Limitation of Liability.**

- a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.
- b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.
- c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

**Section 6 – Term and Termination.**

- a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.
- b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:
  1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or
  2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

- c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.
- d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

**Section 7 – Other Terms and Conditions.**

- a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.
- b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

**Section 8 – Interpretation.**

- a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.
- b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.
- c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.
- d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

# Preformatted

\*Text\*

Table

Table 1: Table

row	row	row
col	col	col
col	col	col
col	col	col

# List of Tables

1	Table . . . . .	12
---	-----------------	----

# List of Figures

## List of References

# Appendix

## @sgrud/bin Module

**@sgrud/bin** - The SGRUD CLI.

### Description

@sgrud/bin - The SGRUD CLI

### Usage

\$ sgrud <command> [options]

### Available Commands

construct	Builds a SGRUD-based project using `microbundle`
kickstart	Kickstarts a SGRUD-based project using `simple-git`
postbuild	Replicates exported package metadata for SGRUD-based projects
runtimify	Creates ESM or UMD bundles for ES6 modules using `microbundle`
universal	Runs SGRUD in universal (SSR) mode using `puppeteer`

For more info, run any command with the `--help` flag

```
$ sgrud construct --help
$ sgrud kickstart --help
```

### Options

-v, --version	Displays current version
-h, --help	Displays this message

### Source

packages/bin/index.ts:1

### bin.

## construct

(Function)

**constructs** a SGRUD-based project using microbundle.

### Description

Constructs a SGRUD-based project using `microbundle`

### Usage

\$ sgrud construct [...modules] [options]

### Options

--compress	Compress/minify build output (default true)
--format	Build specified formats (default commonjs,modern,umd)
--prefix	Use an alternative working directory (default .)
-h, --help	Displays this message

### Examples

```
$ sgrud construct # Run with default options
$ sgrud construct ./project/module # Build ./project/module
$ sgrud construct ./module --format umd # Build ./module as umd
```

### Example

Run with default options:



```
require('@sgrud/bin');
```

```
sgrud.bin.construct();
```

#### Example

```
construct ../project/module:
```

```
require('@sgrud/bin');
```

```
sgrud.bin.construct({
  modules: ['./project/module']
});
```

#### Example

```
construct ../module as umd:
```

```
require('@sgrud/bin');
```

```
sgrud.bin.construct({
  modules: ['./module'],
  format: 'umd'
});
```

#### Signature

```
construct(options?): Promise<void>
```

#### Returns

An execution Promise.

#### Parameters

Name	Type	Default value	Description
options	Object	{}	The options object.
options.compress?	boolean	true	Compress/minify <b>construct</b> output.
options.format?	string	commonjs, modern, umd	<b>construct</b> specified formats.
options.modules?	string[]	undefined	Modules to <b>construct</b> .
options.prefix?	string	./	Use an alternative working directory.

#### Source

packages/bin/src/construct.ts:73

#### bin.

## kickstart

(Function)

**kickstart** a SGRUD-based project using simple-git.

#### Description

Kickstarts a SGRUD-based project using `simple-git`

#### Usage

```
$ sgrud kickstart [library] [options]
```

#### Options

```
--prefix    Use an alternative working directory (default ./)
-h, --help  Displays this message
```

#### Examples

```
$ sgrud kickstart # Run with default options
$ sgrud kickstart preact --prefix ./module # Kickstart preact in ./module
```

#### Example

Run with default options:

```
require('@sgrud/bin');
```

```
sgrud.bin.kickstart();
```

#### Example

**kickstart** preact in ./module:

```
require('@sgrud/bin');
```

```
sgrud.bin.kickstart({
  prefix: './module',
  library: 'preact'
});
```

#### Signature

kickstart(options?): Promise<void>

#### Returns

An execution Promise.

#### Parameters

Name	Type	Default value	Description
options	Object	{}	The options object.
options.library?	string	sgrud	Library which to base upon.
options.prefix?	string	./	Use an alternative working directory.

#### Source

packages/bin/src/kickstart.ts:55

#### bin.

## postbuild

(Function)

Replicates exported package metadata for SGRUD-based projects.

#### Description

Replicates exported package metadata for SGRUD-based projects

#### Usage

```
$ sgrud postbuild [...modules] [options]
```

#### Options

```
--prefix    Use an alternative working directory (default ./)
-h, --help  Displays this message
```

#### Examples

```
$ sgrud postbuild # Run with default options
$ sgrud postbuild ./project/module # Postbuild ./project/module
$ sgrud postbuild --prefix ./module # Run in ./module
```

#### Example

Run with default options:

```
require('@sgrud/bin');
```

```
sgrud.bin.postbuild();
```

#### Example

**postbuild** ./project/module:

```
require('@sgrud/bin');
```

```
sgrud.bin.postbuild({
  modules: ['./project/module']
});
```

**Example**

Run in ./module:

```
require('@sgrud/bin');

sgrud.bin.postbuild({
  prefix: './module'
});
```

**Signature**

postbuild(options?): Promise<void>

**Returns**

An execution Promise.

**Parameters**

Name	Type	Default value	Description
options	Object	{}	The options object.
options.modules?	string[]	undefined	Modules to <b>postbuild</b> .
options.prefix?	string	./	Use an alternative working directory.

**Source**

packages/bin/src/postbuild.ts:67

**bin.****runtimify**

(Function)

Creates ESM or UMD bundles for node modules using microbundle.

**Description**

Creates ESM or UMD bundles for node modules using `microbundle`

**Usage**

```
$ sgrud runtimize [...modules] [options]
```

**Options**

```
--format      Runtimize bundle format (umd or esm) (default umd)
--output      Output file in module root (default runtimize.[format].js)
--prefix      Use an alternative working directory (default ./)
-h, --help    Displays this message
```

**Examples**

```
$ sgrud runtimize # Run with default options
$ sgrud runtimize @microsoft/fast # Runtimize `@microsoft/fast`
```

**Example**

Run with default options (not recommended):

```
require('@sgrud/bin');

sgrud.bin.runtimize();
```

**Example**

**runtimize** @microsoft/fast:

```
require('@sgrud/bin');

sgrud.bin.runtimize({
  modules: ['@microsoft/fast']
});
```

**Signature**

runtimize(options?): Promise<void>

**Returns**

An execution Promise.

**Parameters**

Name	Type	Default value	Description
options	Object	{}	The options object.
options.format?	string	umd	<b>runtimeify</b> bundle format (umd or esm).
options.modules?	string[]	undefined	Modules to <b>runtimeify</b> .
options.output?	string	runtimeify.[format].js	Output file in module root.
options.prefix?	string	./	Use an alternative working directory.

**Source**

packages/bin/src/runtimeify.ts:60

**bin.****universal**

(Function)

Runs SGRUD in **universal** (SSR) mode using puppeteer.

**Description**

Runs SGRUD in universal (SSR) mode using `puppeteer`

**Usage**

```
$ sgrud universal [entry] [options]
```

**Options**

```
--chrome    Chrome executable path (default /usr/bin/chromium-browser)
--prefix     Use an alternative working directory (default ./)
-H, --host   Host/IP to bind to (default 127.0.0.1)
-p, --port   Port to bind to (default 4000)
-h, --help   Displays this message
```

**Examples**

```
$ sgrud universal # Run with default options
$ sgrud universal --host 0.0.0.0 # Listen on all IPs
$ sgrud universal -H 192.168.0.10 -p 4040 # Listen on 192.168.0.10:4040
```

**Example**

Run with default options:

```
require('@sgrud/bin');

sgrud.bin.universal();
```

**Example**

Listen on all IPs:

```
require('@sgrud/bin');

sgrud.bin.universal({
  host: '0.0.0.0'
});
```

**Example**

Listen on 192.168.0.10:4040:

```
require('@sgrud/bin');

sgrud.bin.universal({
  host: '192.168.0.10',
  port: '4040'
});
```

**Signature**

```
universal(options?): Promise<void>
```

**Returns**

An execution Promise.

**Parameters**

Name	Type	Default value	Description
options	Object	{}	The options object.
options.chrome?	string	/usr/bin/chromium-browser	Chrome executable path.
options.entry?	string	index.html	HTML document (relative to prefix).
options.host?	string	127.0.0.1	Host/IP to bind to.
options.port?	string	4000	Port to bind to.
options.prefix?	string	./	Use an alternative working directory.

**Source**

```
packages/bin/src/universal.ts:74
```

## @sgrud/bus Module

**@sgrud/bus** - The SGRUD Software Bus.

The functions and classes found within the **@sgrud/bus** module are intended to ease the internal and external real-time communication of applications building upon the SGRUD client libraries. By establishing a Bus between different modules of an application or between the core of an application and plugins extending it, or even between different applications, loose coupling and data transferral can be achieved.

The **@sgrud/bus** module includes a standalone JavaScript bundle which is used to Spawn a background Thread upon import of this module. This background Thread is henceforth used as central hub for data exchange. Depending on the runtime environment, either a new `worker()` or a new `require('worker_threads').Worker()` NodeJS equivalent will be Spawned.

**Source**

```
packages/bus/index.ts:1
```

**bus.**

## Bus

(Class)

The **Bus** class presents an easy way to establish duplex streams. Through the on-construction supplied Handle the mount point of the created duplex streaming **Bus** within the hierarchical structure of streams handled by the **BusHandler** is designated. Thereby, all Values emitted by the created **Bus** originate from streams beneath the supplied Handle and when invoking the next method of the implemented Observer contract, the resulting Value will originate from this supplied Handle.

An instantiated **Bus** allows for two modes of observation to facilitate simple and complex use cases. The implemented Subscribable contract allows for observation of the dematerialized Values, while the well-known `Symbol.observable` method provides a way to observe the raw Values, including their originating Handles.

**Example**

Using a duplex streaming **Bus**:

```
import { Bus } from '@sgrud/bus';

const bus = new Bus<string, string>('io.github.sgrud.example');

bus.subscribe({ next: console.log });
bus.next('value');
bus.complete();
```

**Type parameters**

Name	Description
I	The input value type of a <b>Bus</b> instance.
O	The output value type of a <b>Bus</b> instance.

**Implements**

Observer<I>, Subscribable<O>

**Source**

packages/bus/src/bus/bus.ts:14, packages/bus/src/bus/bus.ts:109

**bus.Bus.****[observable]**

(Method)

Well-known Symbol.observable method returning a Subscribable. The returned Subscribable emits the raw Values observed by this Bus. By comparison, the implemented subscribe method of the Subscribable interface dematerializes these raw Values before passing them through to the Observer.

**Example**

Subscribe to a raw Bus:

```
import { Bus } from '@sgrud/bus';
import { from } from 'rxjs';

const bus = new Bus<string, string>('io.github.sgrud.example');
from(bus).subscribe(console.log);
```

**Signature**

[observable](): Subscribable<Value<O>>

**Returns**

A Subscribable emitting raw Values.

**Source**

packages/bus/src/bus/bus.ts:179

**bus.Bus.****complete**

(Method)

Implemented **complete** method of the Observer contract. Invoking this method will mark the publishing side of this duplex Bus as **completed**.

**Example**

**complete** a Bus:

```
import { Bus } from '@sgrud/bus';

const bus = new Bus<string, string>('io.github.sgrud.example');
bus.complete();
```

**Signature**

complete(): void

**Source**

packages/bus/src/bus/bus.ts:197

**bus.Bus.****constructor**

(Constructor)

Public Bus **constructor**. The Handle supplied to this **constructor** is assigned as readonlY on the constructed Bus instance and will be used to determine the mount point of this duplex stream within the hierarchical structure of streams handled by the BusHandler.

**Signature**

new Bus<I, O>(handle)

**Type parameters**

Name	Description
I	The input value type of a <b>Bus</b> instance.
O	The output value type of a <b>Bus</b> instance.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to publish this Bus under.

**Source**

packages/bus/src/bus/bus.ts:136

**bus.Bus.****error**

*(Method)*

Implemented **error** method of the Observer contract. Invoking this method will throw the supplied error on the publishing side of this duplex Bus.

**Example**

Throw an **error** through a Bus:

```
import { Bus } from '@sgrud/bus';

const bus = new Bus<string, string>('io.github.sgrud.example');
bus.error(new Error('example'));
```

**Signature**

error(error): void

**Parameters**

Name	Type	Description
error	unknown	The error to publish.

**Source**

packages/bus/src/bus/bus.ts:217

**bus.Bus.****handle**

*(Readonly Property)*

The Handle to publish this Bus under.

**Source**

packages/bus/src/bus/bus.ts:141

**bus.Bus.****next**

*(Method)*

Implemented **next** method of the Observer contract. Invoking this method will provide any observer of the publishing side of this duplex Bus with the **next** value.

**Example**

Supplying a Bus with a **next** value:

```
import { Bus } from '@sgrud/bus';

const bus = new Bus<string, string>('io.github.sgrud.example');
bus.next('value');
```

**Signature**

next(value): void

**Parameters**

Name	Type	Description
value	I	The <b>next</b> value to publish.

**Source**

packages/bus/src/bus/bus.ts:237

**bus.Bus.****subscribe**

(Method)

Implemented **subscribe** method of the Subscribable contract. Invoking this method while supplying an observer will **subscribe** the supplied observer to any changes on the observed side of this duplex Bus.

**Example**

**subscribe** to a dematerialized Bus:

```
import { Bus } from '@sgrud/bus';

const bus = new Bus<string, string>('io.github.sgrud.example');
bus.subscribe({ next: console.log });
```

**Signature**

subscribe(observer?): Unsubscribable

**Returns**

An Unsubscribable of the ongoing observation.

**Parameters**

Name	Type	Description
observer?	Partial<Observer<0>>	The observer to <b>subscribe</b> to this Bus.

**Source**

packages/bus/src/bus/bus.ts:259

**bus.Bus.****observe**

(Private Readonly Property)

The **observed** side of this Bus. The Observable assigned to this property is used to fulfill the Subscribable contract and is obtained through the BusHandler.

**Source**

packages/bus/src/bus/bus.ts:116

**bus.Bus.****publish**

(Private Readonly Property)

The **publishing** side of this Bus. The Subject assigned to this property is used to fulfill the Observer contract and is provided to the BusHandler for **publishment**.



**Source**

packages/bus/src/bus/bus.ts:123

**bus.****Bus**

*(Namespace)*

The **Bus** namespace contains types and interfaces used and intended to be used in conjunction with the Singleton BusHandler class. This namespace contains the Handle string literal type helper, designating the hierarchical mount-point of any **Bus**, as well as the Value type helper, describing the data and state a **Bus** may transport.

**See**

Bus

**Source**

packages/bus/src/bus/bus.ts:14, packages/bus/src/bus/bus.ts:109

**bus.Bus.****Handle**

*(Type alias)*

The **Handle** string literal helper type enforces any assigned value to contain at least three dots. It represents a type constraint which should be thought of as domain name in reverse notation. All employed **Handles** thereby designate a hierarchical structure, which the BusHandler in conjunction with the BusWorker operate upon.

**Example**

Library-wide **Handle**:

```
import { type Bus } from '@sgrud/bus';

const busHandle: Bus.Handle = 'io.github.sgrud';
```

**Example**

An invalid **Handle**:

```
import { type Bus } from '@sgrud/bus';

const busHandle: Bus.Handle = 'org.example';
// Type [...] is not assignable to type 'Handle'.
```

**See**

BusHandler

**Source**

packages/bus/src/bus/bus.ts:42

**bus.Bus.****Value**

*(Type alias)*

The **Value** type helper extends the ObservableNotification type and describes the shape of all values emitted by any stream handled by the BusHandler. As those streams are Observables, which are dynamically combined through their hierarchical structure denoted by their corresponding Handles and therefore may emit from more than one Handle, each **Value** emitted by any bus contains its originating Handle.

**Example**

Logging emitted **Values**:

```
import { BusHandler } from '@sgrud/bus';

const busHandler = new BusHandler();
busHandler.observe('io.github.sgrud').subscribe(console.log);
// { handle: 'io.github.sgrud.example', type: 'N', value: 'published' }
```

**See**

BusHandler

**Type parameters**

Name	Description
T	The Bus <b>Value</b> type.

**Source**

packages/bus/src/bus/bus.ts:67

**bus.**

**BusHandler**

(Class)

The **BusHandler** implements and orchestrates the establishment, transferral and deconstruction of any number of Observable streams. It operates in conjunction with the BusWorker Thread which is run in the background. To designate and organize different Observable streams, the string literal helper type Handle is employed. As an example, let the following hierarchical structure be given:

```
io.github.sgrud
├─ io.github.sgrud.core
│   └─ io.github.sgrud.core.kernel
│       └─ io.github.sgrud.core.transit
├─ io.github.sgrud.data
│   └─ io.github.sgrud.data.model.current
│       └─ io.github.sgrud.data.model.global
├─ io.github.sgrud.shell
│   └─ io.github.sgrud.shell.route
├─ io.github.sgrud.store
│   └─ io.github.sgrud.store.global
│       └─ io.github.sgrud.store.local
```

Depending on the Handle, one may observe all established streams beneath the root `io.github.sgrud` Handle or only one specific stream, e.g., `io.github.sgrud.core.kernel`. The Observable returned from the `observe` method will emit all Values originating from all streams beneath the root Handle in the first case, or only Values from one stream, in the second case.

**Decorator**

Singleton

**See**

BusWorker

**Source**

packages/bus/src/handler/handler.ts:46

**bus.BusHandler.**

**[observable]**

(Static Method)

Static `Symbol.observable` method returning a `Subscribable`. The returned `Subscribable` mirrors the private loader and is used for initializations after the `BusHandler` has been successfully initialized.

**Example**

Subscribe to the `BusHandler`:

```
import { BusHandler } from '@sgrud/bus';
import { from } from 'rxjs';

from(BusHandler).subscribe(console.log);
```

**Signature**

```
[observable](): Subscribable<BusHandler>
```

**Returns**

A `Subscribable` emitting this `BusHandler`.

**Source**

packages/bus/src/handler/handler.ts:72

**bus.BusHandler.****loader**

(Static Private Property)

Private static ReplaySubject used as the BusHandler **loader**. This **loader** emits once after the BusHandler has been successfully initialized.

**Source**

packages/bus/src/handler/handler.ts:53

**bus.BusHandler.****constructor**

(Constructor)

Public BusHandler **constructor**. As the BusHandler is a Singleton class, this **constructor** is only invoked the first time it is targeted by the new operator. Upon this first invocation, the worker property is assigned an instance of the BusWorker Thread while using the supplied source, if any.

**Throws**

A ReferenceError when the environment is incompatible.

**Signature**

```
new BusHandler(source?)
```

**Parameters**

Name	Type	Description
source?	string	An optional Module source.

**Source**

packages/bus/src/handler/handler.ts:104

**bus.BusHandler.****observe**

(Method)

Invoking this method **observes** the Observable stream represented by the supplied handle. The method will return an Observable originating from the BusWorker which emits all Values published under the supplied handle. When the **observe** method is invoked with 'io.github.sgrud', all streams hierarchically beneath this Handle, e.g., 'io.github.bus.status', will also be emitted by the returned Observable.

**Example**

**observe** the 'io.github.sgrud' stream:

```
import { BusHandler } from '@sgrud/bus';

const busHandler = new BusHandler();
const handle = 'io.github.sgrud.example';

busHandler.observe(handle).subscribe(console.log);
```

**Signature**

```
observe<T>(handle): Observable<Value<T>>
```

**Returns**

An Observable bus for handle.

**Type parameters**

Name	Description
T	The type of the <b>observed</b> Observable stream.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>observe</b> .

**Source**

packages/bus/src/handler/handler.ts:160

**bus.BusHandler.****publish**

(Method)

Invoking this method **publishes** the supplied Observable stream under the supplied handle. This method returns an Observable of the **publishment** of the supplied Observable stream under the supplied handle with the BusWorker. When the **published** source Observable completes, the registration within the BusWorker will automatically self-destruct.

**Example**

**publish** a stream under 'io.github.sgrud.example':

```
import { BusHandler } from '@sgrud/bus';
import { of } from 'rxjs';

const busHandler = new BusHandler();
const handle = 'io.github.sgrud.example';
const stream = of('published');

busHandler.publish(handle, stream).subscribe();
```

**Signature**

publish<T>(handle, stream): Observable<void>

**Returns**

An Observable of the stream **publishment**.

**Type parameters**

Name	Description
T	The type of the <b>published</b> Observable stream.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>publish</b> under.
stream	ObservableInput<T>	The Observable stream for handle.

**Source**

packages/bus/src/handler/handler.ts:193

**bus.BusHandler.****uplink**

(Method)

Invoking this method **uplinks** the supplied handle to the supplied url by establishing a WebSocket connection between the endpoint behind the supplied url and the BusWorker. This method returns an Observable of the **uplink** Subscription which can be used to cancel the **uplink**. When the **uplinked** WebSocket is closed or throws an error, it is automatically cleaned up and unsubscribed from.

**Example**

**uplink** the 'io.github.sgrud.uplink' Handle:

```
import { BusHandler } from '@sgrud/bus';

const busHandler = new BusHandler();
const handle = 'io.github.sgrud.example';
const url = 'https://example.com/websocket';

const uplink = busHandler.uplink(handle, url).subscribe();
```

**Signature**

uplink(handle, url): Observable<Subscription>

**Returns**

An Observable of the **uplink** Subscription.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>uplink</b> .
url	string	The endpoint url to establish an <b>uplink</b> to.

**Source**

packages/bus/src/handler/handler.ts:231

**bus.BusHandler.****worker**

*(Readonly Property)*

The **worker** Thread and main background workhorse. The underlying BusWorker is run inside a Worker context in the background and transparently handles published and observed streams and the aggregation of their values depending on their Handle, i.e., hierarchy.

**See**

BusWorker

**Source**

packages/bus/src/handler/handler.ts:92

**bus.****BusQuerier**

*(Class)*

The **BusQuerier** implements an Bus based Querier, i.e., extension of the abstract Querier base class, allowing Model queries to be executed via a Bus. To use this class, provide it to the Linker by either extending it, and decorating the extending class with the Target decorator, or by preemptively supplying an instance of this class to the Linker.

**Example**

Provide the **BusQuerier** to the Linker:

```
import { BusQuerier } from '@sgrud/bus';
import { Linker } from '@sgrud/core';

new Linker<typeof BusQuerier>([
  [BusQuerier, new BusQuerier('io.github.sgrud.example')]
]);
```

**See**

Model, Querier

**Hierarchy**

- Querier<this>
  - **BusQuerier**

**Source**

packages/bus/src/bus/querier.ts:28

**bus.BusQuerier.****[provide]**

(Static Readonly Property)

Magic string by which this class is provided.

**See**

provide

**Source**

packages/data/src/querier/querier.ts:96

**bus.BusQuerier.****commit**

(Method)

Overridden **commit** method of the Querier base class. When this Querier is made available via the Linker, this overridden **commit** method is called when this Querier claims the highest priority to **commit** an Operation, depending on the Model from which the Operation originates.

**Signature**

commit(operation, variables): Observable<unknown>

**Returns**

An Observable of the **committed** operation.

**Parameters**

Name	Type	Description
operation	Operation	The Operation to be <b>committed</b> .
variables	Variables	Any Variables within the operation.

**Source**

packages/bus/src/bus/querier.ts:82

**bus.BusQuerier.****constructor**

(Constructor)

Public **constructor** consuming the handle Model queries should be committed through, and an dynamic or static prioritize value. The prioritize value may either be a mapping of Models to corresponding priorities or a static priority for this Querier.

**Signature**

new BusQuerier(handle, prioritize?)

**Parameters**

Name	Type	Default value	Description
handle	Handle	undefined	The Handle to commit queries under.
prioritize	number   Map<Type<Model<any>>, number>	0	The dynamic or static prioritization.

**Source**

packages/bus/src/bus/querier.ts:51

**bus.BusQuerier.****priority***(Method)*

Overridden **priority** method of the Querier base class. When an Operation is to be committed, this method is called with the respective `model` Type and returns the claimed **priority** to commit this Model.

**Signature**

```
priority(model): number
```

**Returns**

The numeric **priority** of this Querier implementation.

**Parameters**

Name	Type	Description
<code>model</code>	<code>Type&lt;Model&lt;any&gt;&gt;</code>	The Model to be committed.

**Source**

```
packages/bus/src/bus/querier.ts:107
```

**bus.BusQuerier.****types***(Readonly Property)*

A set containing the Types this BusQuerier handles. As a Bus is a long-lived duplex stream, this Querier can handle 'mutation', 'query' and 'subscription' **types**.

**Source**

```
packages/bus/src/bus/querier.ts:36
```

**bus.BusQuerier.****handle***(Private Readonly Property)*

The Handle to commit queries under.

**Source**

```
packages/bus/src/bus/querier.ts:56
```

**bus.BusQuerier.****prioritize***(Private Readonly Property)*

The dynamic or static prioritization.

**See**

`priority`

**Source**

```
packages/bus/src/bus/querier.ts:65
```

**bus.****BusWorker***(Class)*

The **BusWorker** is a background Thread which is Spawned by the BusHandler to handle all published and observed streams, uplinks and their aggregation depending on their hierarchy.

**Decorator**

Thread, Singleton

**See**

BusHandler

**Source**

packages/bus/src/worker/index.ts:24

**bus.BusWorker.****constructor**

*(Constructor)*

Public **constructor**. This **constructor** is called once when the BusHandler Spawns this BusWorker.

**Remarks**

This method should only be invoked by the BusHandler.

**Signature**

```
new BusWorker()
```

**Source**

packages/bus/src/worker/index.ts:52

**bus.BusWorker.****observe**

*(Method)*

Invoking this method **observes** all Observable streams under the supplied `handle` by mergeing all streams which are published under the supplied `handle`.

**Remarks**

This method should only be invoked by the BusHandler.

**Signature**

```
observe<T>(handle): Promise<Observable<Value<T>>>
```

**Returns**

An Observable stream for `handle`.

**Type parameters**

Name	Description
T	The type of the <b>observed</b> Observable stream.

**Parameters**

Name	Type	Description
<code>handle</code>	Handle	The Handle to <b>observe</b> .

**Source**

packages/bus/src/worker/index.ts:69

**bus.BusWorker.****publish**

*(Method)*

Invoking this method **publishes** the supplied ObservableInput stream under the supplied `handle`. Any emittance of the **published** stream will be materialized into Values and replayed once to every observer.

**Throws**

A ReferenceError on collision of `handles`.



**Remarks**

This method should only be invoked by the BusHandler.

**Signature**

```
publish<T>(handle, stream): Promise<void>
```

**Returns**

A Promise of the stream **publishment**.

**Type parameters**

Name	Description
T	The type of the <b>published</b> Observable stream.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>publish</b> under.
stream	ObservableInput<T>	The ObservableInput stream for handle.

**Source**

packages/bus/src/worker/index.ts:113

**bus.BusWorker.****uplink**

(Method)

Invoking this method **uplinks** the supplied handle to the supplied url by establishing a WebSocket connection between the endpoint behind the supplied url and this BusWorker. It is assumed, that all messages emanating from the WebSocket endpoint conform to the Value type and are therefore treated as such. This treatment includes the filtering of all received and submitted messages by comparing their corresponding Handle and the supplied handle.

**Throws**

A ReferenceError on collision of handles.

**Remarks**

This method should only be invoked by the BusHandler.

**Signature**

```
uplink(handle, url): Promise<Subscription>
```

**Returns**

A Promise of the Subscription to the **uplink**.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>uplink</b> .
url	string	The endpoint url to establish an <b>uplink</b> to.

**Source**

packages/bus/src/worker/index.ts:152

**bus.BusWorker.****changes**

(Private Readonly Property)

BehaviorSubject emitting every time when **changes** occur on the internal streams or uplinks mappings. This emittance is used to recombine the Observable streams which were previously obtained to through use of the observe method.

**Source**

packages/bus/src/worker/index.ts:32

**bus.BusWorker.****streams**

*(Private Readonly Property)*

Internal Mapping containing all established **streams**. Updating this map should always be accompanied by an emittance of changes.

**Source**

packages/bus/src/worker/index.ts:38

**bus.BusWorker.****uplinks**

*(Private Readonly Property)*

Internal Mapping containing all established **uplinks**. Updating this map should always be accompanied by an emittance of changes.

**Source**

packages/bus/src/worker/index.ts:44

**bus.****Observe**

*(Function)*

Prototype property decorator factory. Applying this decorator replaces the decorated property with a getter returning an Observable stream which **Observes** all values originating from the supplied `handle`. Depending on the value of the `suffix` parameter, this Observable stream is either assigned directly to the prototype using the supplied `handle`, or, if a truthy value is supplied for the `suffix` parameter, this value is assumed to reference another property of the class containing this decorated property. The first truthy value assigned to this `suffix` property on an instance of the class containing this **Stream** decorator will then be used to suffix the supplied `handle` which is to be **Observed** and assign the resulting Observable stream to the decorated instance property.

This decorator is more or less the opposite of the Publish decorator, while both rely on the BusHandler to fulfill contracts.

**Example**

Observe the 'io.github.sgrud.example' stream:

```
import { type Bus, Observe } from '@sgrud/bus';
import { type Observable } from 'rxjs';

export class Observer {
  @Observe('io.github.sgrud.example')
  public readonly stream!: Observable<Bus.Value<unknown>>;
}

Observer.prototype.stream.subscribe(console.log);
```

**Example**

Observe the 'io.github.sgrud.example' stream:

```
import { type Bus, Observe } from '@sgrud/bus';
import { type Observable } from 'rxjs';

export class Observer {
  @Observe('io.github.sgrud', 'suffix')
  public readonly stream!: Observable<Bus.Value<unknown>>;

  public constructor(
    public readonly suffix: string
  ) { }
}
```

```
const observer = new Observer('example');
observer.stream.subscribe(console.log);
```

**See**

BusHandler, Publish, Stream

**Signature**

Observe(handle, suffix?): (prototype: object, propertyKey: PropertyKey) => void

**Returns**

A prototype property decorator.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>Observe</b> .
suffix?	PropertyKey	An optional suffix property for the handle.

**Source**

packages/bus/src/handler/observe.ts:66

**bus.****Publish**

(Function)

Prototype property decorator factory. This decorator **Publishes** a newly instantiated Subject under the supplied handle and assigns it to the decorated property. Depending on the value of the suffix parameter, this newly instantiated Subject is either assigned directly to the prototype and **Published** using the supplied handle, or, if a truthy value is supplied for the suffix parameter, this value is assumed to reference another property of the class containing this decorated property. The first truthy value assigned to this suffix property on an instance of the class containing this **Publish** decorator will then be used to suffix the supplied handle upon **Publishment** of the newly instantiated Subject, which is assigned to the decorated instance property.

Through these two different modes of operation, the Subject that will be **Published** can be assigned statically to the prototype of the class containing the decorated property, or this assignment can be deferred until an instance of the class containing the decorated property is constructed and a truthy value is assigned to its suffix property.

This decorator is more or less the opposite of the Observe decorator, while both rely on the BusHandler to fulfill contracts. Furthermore, precautions should be taken to ensure the completion of the **Published** Subject as memory leaks may occur due to dangling subscriptions.

**Example**

**Publish** the 'io.github.sgrud.example' stream:

```
import { Publish } from '@sgrud/bus';
import { type Subject } from 'rxjs';

export class Publisher {

  @Publish('io.github.sgrud.example')
  public readonly stream!: Subject<unknown>;

}
```

```
Publisher.prototype.stream.next('value');
Publisher.prototype.stream.complete();
```

**Example**

**Publish** the 'io.github.sgrud.example' stream:

```
import { Publish } from '@sgrud/bus';
import { type Subject } from 'rxjs';

export class Publisher {

  @Publish('io.github.sgrud', 'suffix')
  public readonly stream: Subject<unknown>;

  public constructor(
```

```

    private readonly suffix: string
  ) {}
}

const publisher = new Publisher('example');
publisher.stream.next('value');
publisher.stream.complete();

```

**See**

BusHandler, Observe, Stream

**Signature**

Publish(handle, suffix?): (prototype: object, propertyKey: PropertyKey) => void

**Returns**

A prototype property decorator.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>Publish</b> .
suffix?	PropertyKey	An optional suffix property for the handle.

**Source**

packages/bus/src/handler/publish.ts:76

**bus.****Stream**

(Function)

Prototype property decorator factory. Applying this decorator replaces the decorated property with a getter returning a Bus, thereby allowing to duplex **Stream** values designated by the supplied handle. Depending on the value of the suffix parameter, this Bus is either assigned directly to the prototype using the supplied handle, or, if a truthy value is supplied for the suffix parameter, this value is assumed to reference another property of the class containing this decorated property. The first truthy value assigned to this suffix property on an instance of the class containing this **Stream** decorator will then be used to suffix the supplied handle upon construction of the Bus, which is assigned to the decorated instance property.

Through these two different modes of operation, a Bus can be assigned statically to the prototype of the class containing the decorated property, or this assignment can be deferred until an instance of the class containing the decorated property is constructed and a truthy value is assigned to its suffix property.

**Example**

**Stream** 'io.github.sgrud.example':

```

import { type Bus, Stream } from '@sgrud/bus';

export class Streamer {
  @Stream('io.github.sgrud.example')
  public readonly stream!: Bus<unknown, unknown>;
}

Streamer.prototype.stream.next('value');
Streamer.prototype.stream.complete();

Streamer.prototype.stream.subscribe({
  next: console.log
});

```

**Example**

**Stream** 'io.github.sgrud.example':

```

import { type Bus, Stream } from '@sgrud/bus';

```

```
export class Streamer {

  @Stream('io.github.sgrud', 'suffix')
  public readonly stream!: Bus<unknown>;

  public constructor(
    public readonly suffix: string
  ) { }

}

const streamer = new Streamer('example');
streamer.stream.next('value');
streamer.stream.complete();

streamer.stream.subscribe({
  next: console.log
});
```

**See**

BusHandler, Observe, Publish

**Signature**

Stream(handle, suffix?): (prototype: object, propertyKey: PropertyKey) => void

**Returns**

A prototype property decorator.

**Parameters**

Name	Type	Description
handle	Handle	The Handle to <b>Stream</b> .
suffix?	PropertyKey	An optional suffix property for the handle.

**Source**

packages/bus/src/handler/stream.ts:75

## @sgrud/core Module

**@sgrud/core** - The SGRUD Core Module.

The functions and classes found within the **@sgrud/core** module represent the base upon which the SGRUD client libraries are built. Therefore, most of the code provided within this module does not aim at fulfilling one specific high-level need, but is used and intended to be used as low-level building blocks for downstream projects. This practice is employed throughout the SGRUD client libraries, as all modules depend on this core module. By providing the core functionality within this singular module, all downstream SGRUD modules should be considered opt-in functionality which may be used within projects building upon the SGRUD client libraries.

**Source**

packages/core/index.ts:1

**core.**

### Alias

(Type alias)

Type helper **Aliasing** any provided Type. By looping a Type through this **Alias** type helper, the dereferencing of this Type is prohibited. Use this helper to, e.g., force a string literal type to be treated as an unique type and not to be dereferenced.

**Example**

**Alias** the `${number} ${'<' | '>'} ${number}` type:

```
import { type Alias } from '@sgrud/core';

type Helper = Alias<`${number} ${'<' | '>'} ${number}`>;
```

```
const negative: Helper = '-01 < +0.1'; // negative: Helper
const positive: Helper = 'one is > 0'; // not assignable to type 'Helper'
```

**Remarks**

<https://github.com/microsoft/TypeScript/issues/47828>

**Type parameters**

Name	Description
T	The type that should be <b>Aliased</b> .

**Source**

packages/core/src/typing/alias.ts:22

**core.**

## Assign

(Type alias)

Type helper **Assign**ing the own property types of all of the enumerable own properties from a source type to a target type.

**Example**

**Assign** `valueOf()` to `string`:

```
import { type Assign } from '@sgrud/core';

const str = 'Hello world' as Assign<{
  valueOf(): 'Hello world';
}, string>;
```

**Type parameters**

Name	Description
S	The source type to <b>Assign</b> from.
T	The target type to <b>Assign</b> to.

**Source**

packages/core/src/typing/assign.ts:18

**core.**

## Factor

(Function)

Prototype property decorator factory. Applying this decorator replaces the decorated prototype property with a getter, which returns the linked instance of a Targeted constructor, referenced by the `targetFactory`. Depending on the supplied transient value, the target constructor is invoked to construct (and link) an instance, if none is linked beforehand.

**Example**

**Factor** an eager and lazy service:

```
import { Factor } from '@sgrud/core';
import { EagerService, LazyService } from '../services';

export class ServiceHandler {

  @Factor(() => EagerService)
  private readonly service!: EagerService;

  @Factor(() => LazyService, true)
  private readonly service?: LazyService;

}
```

**See**

Linker, Target

**Signature**

Factor<K>(targetFactory, transient?): (prototype: object, propertyKey: PropertyKey) => void

**Returns**

A prototype property decorator.

**Type parameters**

Name	Type	Description
K	extends () => any	The Targeted constructor type.

**Parameters**

Name	Type	Default value	Description
targetFactory	() => K	undefined	A forward reference to the target constructor.
transient	boolean	false	Whether an instance is constructed if none is linked.

**Source**

packages/core/src/linker/factor.ts:35

**core.****Http**

(Abstract Class)

The abstract **Http** class is a thin wrapper around the ajax method. The main function of this wrapper is to pipe all requests through a chain of classes extending the abstract Proxy class. Thereby interceptors for various requests can be implemented to, e.g., provide API credentials etc.

**See**

Proxy

**Implements**

Handler

**Source**

packages/core/src/http/http.ts:13, packages/core/src/http/http.ts:57

**core.Http.****delete**

(Static Method)

Fires an Http **delete** request against the supplied url upon subscription.

**Example**

Fire an HTTP **delete** request against `https://example.com`:

```
import { Http } from '@sgrud/core';

Http.delete('https://example.com').subscribe(console.log);
```

**Signature**

delete<T>(url): Observable<Response<T>>

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
url	string	The url to Http <b>delete</b> .

**Source**

packages/core/src/http/http.ts:75

**core.Http.****get**

*(Static Method)*

Fires an Http **get** request against the supplied url upon subscription.

**Example**

Fire an HTTP **GET** request against `https://example.com`:

```
import { Http } from '@sgrud/core';
Http.get('https://example.com').subscribe(console.log);
```

**Signature**

```
get<T>(url): Observable<Response<T>>
```

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
url	string	The url to Http <b>get</b> .

**Source**

packages/core/src/http/http.ts:95

**core.Http.****head**

*(Static Method)*

Fires an Http **head** request against the supplied url upon subscription.

**Example**

Fire an HTTP **head** request against `https://example.com`:

```
import { Http } from '@sgrud/core';
Http.head('https://example.com').subscribe(console.log);
```



**Signature**

```
head<T>(url): Observable<Response<T>>
```

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
url	string	The url to Http <b>head</b> .

**Source**

packages/core/src/http/http.ts:115

**core.Http.****patch**

(Static Method)

Fires an Http **patch** request against the supplied url containing the supplied body upon subscription.

**Example**

Fire an HTTP **patch** request against https://example.com:

```
import { Http } from '@sgrud/core';

Http.patch('https://example.com', {
  data: 'value'
}).subscribe(console.log);
```

**Signature**

```
patch<T>(url, body): Observable<Response<T>>
```

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
url	string	The url to Http <b>patch</b> .
body	unknown	The body of the Request.

**Source**

packages/core/src/http/http.ts:138

**core.Http.****post**

(Static Method)

Fires an Http **post** request against the supplied url containing the supplied body upon subscription.

**Example**

Fire an HTTP **post** request against `https://example.com`:

```
import { Http } from '@sgrud/core';

Http.post('https://example.com', {
  data: 'value'
}).subscribe(console.log);
```

**Signature**

`post<T>(url, body): Observable<Response<T>>`

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
url	string	The url to Http <b>post</b> .
body	unknown	The body of the Request.

**Source**

`packages/core/src/http/http.ts:164`

**core.Http.****put**

*(Static Method)*

Fires an Http **put** request against the supplied url containing the supplied body upon subscription.

**Example**

Fire an HTTP **put** request against `https://example.com`:

```
import { Http } from '@sgrud/core';

Http.put('https://example.com', {
  data: 'value'
}).subscribe(console.log);
```

**Signature**

`put<T>(url, body): Observable<Response<T>>`

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
url	string	The url to Http <b>put</b> .
body	unknown	The body of the Request.

**Source**

packages/core/src/http/http.ts:190

**core.Http.  
request**

(Static Method)

Fires a custom Request. Use this method for more fine-grained control over the outgoing Request.

**Example**

Fire an HTTP custom request against `https://example.com`:

```
import { Http } from '@sgrud/core';

Http.request({
  method: 'GET',
  url: 'https://example.com',
  headers: { 'x-example': 'value' }
}).subscribe(console.log);
```

**Signature**

`request<T>(request): Observable<Response<T>>`

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The Response type.

**Parameters**

Name	Type	Description
request	Request	The Request to be <b>requested</b> .

**Source**

packages/core/src/http/http.ts:217

**core.Http.  
handle**

(Method)

Generic **handle** method, enforced by the Handler interface. Main method of the this class. Internally pipes the request through all linked classes extending Proxy.

**Signature**

`handle<T>(request): Observable<Response<T>>`

**Returns**

An Observable of the Response.

**Type parameters**

Name	Description
T	The type of the <b>handled</b> Response.

**Parameters**

Name	Type	Description
request	Request	The Request to be <b>handled</b> .

**Source**

packages/core/src/http/http.ts:241

**core.Http.****constructor**

*(Private Constructor)*

Private **constructor** (which should never be called).

**Throws**

A `TypeError` upon construction.

**Signature**

`new Http()`

**Source**

packages/core/src/http/http.ts:228

**core.****Http**

*(Namespace)*

The **Http** namespace contains types and interfaces used and intended to be used in conjunction with the abstract `Http` class.

**See**

`Http`

**Source**

packages/core/src/http/http.ts:13, packages/core/src/http/http.ts:57

**core.Http.****Handler**

*(Interface)*

The **Handler** interface enforces the `handle` method with ajax compliant typing on the implementing class or object. This contract is used by the `Proxy` to type-guard the next hops.

**Implemented by**

`Http`

**Source**

packages/core/src/http/http.ts:34

**core.Http.Handler.****handle**

*(Method)*

Generic **handle** method enforcing ajax compliant typing. The method signature corresponds to that of the ajax method itself.

**Signature**

`handle(request): Observable<Response<any>>`

**Returns**

An `Observable` of the requested `Response`.

**Parameters**

Name	Type	Description
request	Request	Requesting Request.

**Source**

packages/core/src/http/http.ts:43

**core.Http.****Request**

(Type alias)

The **Request** type alias references the AjaxConfig interface and describes the shape of any Http **Request** parameters.

**Source**

packages/core/src/http/http.ts:19

**core.Http.****Response**

(Type alias)

The **Response** type alias references the AjaxResponse class and describes the shape of any Http **Response**.

**Type parameters**

Name	Type	Description
T	any	The <b>Response</b> type of a Request.

**Source**

packages/core/src/http/http.ts:27

**core.****Kernel**

(Class)

Singleton **Kernel** class. The **Kernel** is essentially a dependency loader for ESM bundles (and their respective `importmaps`) or, depending on the runtime context and capabilities, UMD bundles and their transitive dependencies. By making use of the **Kernel**, applications based on the SGRUD client libraries may be comprised of multiple, optionally loaded Modules.

**Decorator**

Singleton

**Source**

packages/core/src/kernel/kernel.ts:16, packages/core/src/kernel/kernel.ts:159

**core.Kernel.****[observable]**

(Method)

Well-known `Symbol.observable` method returning a `Subscribable`. The returned `Subscribable` emits every `Module` that is successfully loaded.

**Example**

Subscribe to the loaded Modules:

```
import { Kernel } from '@sgrud/core';
import { from } from 'rxjs';

from(new Kernel()).subscribe(console.log);
```

**Signature**

```
[observable](): Subscribable<Module>
```

**Returns**

A Subscribable emitting loaded Modules.

**Source**

packages/core/src/kernel/kernel.ts:265

**core.Kernel.****constructor**

(Constructor)

Singleton **constructor**. The first time, this **constructor** is called, it will persist the nodeModules path Modules should be loaded from. Subsequent **constructor** calls will ignore this argument and return the Singleton instance. Through subscribing to the Subscribable returned by the well-known Symbol.observable method, the Module loading progress can be tracked.

**Example**

Instantiate the **Kernel** and require Modules:

```
import { Kernel } from '@sgrud/core';
import { forkJoin } from 'rxjs';

const kernel = new Kernel('https://unpkg.com');

forkJoin([
  kernel.require('example-module'),
  kernel.require('/static/local-module')
]).subscribe(console.log);
```

**Signature**

```
new Kernel(nodeModules?)
```

**Parameters**

Name	Type	Default value	Description
nodeModules	string	' /node_modules '	Optional location to load node modules from.

**Source**

packages/core/src/kernel/kernel.ts:221

**core.Kernel.****insmod**

(Method)

Calling this method while supplying a valid module definition will chain the **insert module** operations of the module dependencies and the module itself into an Observable, which is then returned. When multiple Modules are inserted, their dependencies are deduplicated by internally tracking all Modules and their transitive dependencies as separate loaders. Depending on the browser context, either the UMD or ESM bundles (and their respective import maps) are loaded via calling the script method. When **insmodding** Modules which contain transitive sgrudDependencies, their compatibility is checked. Should a dependency version mismatch, the Observable returned by this method will throw.

**Throws**

An Observable RangeError or ReferenceError.

**Example**

**insmod** a Module by definition:

```
import { Kernel } from '@sgrud/core';
import packageJson from './module/package.json';

new Kernel().insmod(packageJson).subscribe(console.log);
```

**Signature**

```
insmod(module, source?, execute?): Observable<Module>
```

**Returns**

An Observable of the Module definition.

**Parameters**

Name	Type	Default value	Description
module	Module	undefined	The Module definition to <b>insmod</b> .
source	string	undefined	An optional Module source.
execute	boolean	false	Whether to execute the Module.

**Source**

packages/core/src/kernel/kernel.ts:298

**core.Kernel.****nodeModules**

*(Readonly Property)*

Optional location to load node modules from.

**Source**

packages/core/src/kernel/kernel.ts:228

**core.Kernel.****require**

*(Method)*

**requires** a Module by name or source. If the supplied `id` is a relative path starting with `./`, an absolute path starting with `/` or an URL starting with `http`, the `id` is used as-is, otherwise it is appended to the `nodeModules` path and the `package.json` file within this path is retrieved via Http GET. The Module definition is then passed to the `insmod` method and returned.

**Example**

**require** a Module by `id`:

```
import { Kernel } from '@sgrud/core';

new Kernel().require('/static/lazy-module').subscribe(console.log);
```

**Signature**

```
require(id, execute?): Observable<Module>
```

**Returns**

An Observable of the Module definition.

**Parameters**

Name	Type	Default value	Description
id	string	undefined	The Module name or source to <b>require</b> .
execute	boolean	true	Whether to execute the Module.

**Source**

packages/core/src/kernel/kernel.ts:427

**core.Kernel.****resolve***(Method)*

**resolves** a Module definition by its name. The Module name is appended to the source path or, if none is supplied, the nodeModules path and the package.json file therein retrieved via Http GET. The parsed package.json is then emitted by the returned Observable.

**Example**

**resolve** a Module definition:

```
import { Kernel } from '@sgrud/core';

new Kernel().resolve('module').subscribe(console.log);
```

**Signature**

resolve(name, source?): Observable<Module>

**Returns**

An Observable of the Module definition.

**Parameters**

Name	Type	Description
name	string	The Module name to <b>resolve</b> .
source	string	An optional Module source.

**Source**

packages/core/src/kernel/kernel.ts:461

**core.Kernel.****script***(Method)*

Inserts an HTMLScriptElement and applies the supplied props to it. The returned Observable emits and completes when the on load handler of the HTMLScriptElement is called. If no external src is supplied through the props, the on load handler of the element is called asynchronously. When the returned Observable completes, the inserted HTMLScriptElement is removed.

**Example**

Insert an HTMLScriptElement:

```
import { Kernel } from '@sgrud/core';

new Kernel().script({
  src: '/node_modules/module/bundle.js',
  type: 'text/javascript'
}).subscribe();
```

**Signature**

script(props): Observable<void>

**Returns**

An Observable of the HTMLScriptElements on load.

**Parameters**

Name	Type	Description
props	Partial<HTMLScriptElement>	Any properties to apply to the HTMLScriptElement.

**Source**

packages/core/src/kernel/kernel.ts:498



**core.Kernel.****verify***(Method)*

Inserts an HTMLLinkElement and applies the supplied props to it. This method is used to **verify** a Module bundle before importing and executing it by **verifying** its Digest.

**Example**

**verify** a Module by Digest:

```
import { Kernel } from '@sgrud/core';

new Kernel().verify({
  href: '/node_modules/module/index.js',
  integrity: 'sha256- [...]',
  rel: 'modulepreload'
}).subscribe();
```

**Signature**

verify(props): Observable<void>

**Returns**

An Observable of the appendage and removal of the element.

**Parameters**

Name	Type	Description
props	Partial<HTMLLinkElement>	Any properties to apply to the HTMLLinkElement.

**Source**

packages/core/src/kernel/kernel.ts:539

**core.Kernel.****changes***(Private Readonly Property)*

Internal ReplaySubject tracking the loading state and therefore **changes** of loaded Modules. An Observable form of this internal ReplaySubject may be retrieved by invoking the well-known Symbol.observable method and subscribing to the returned Subscribable. The internal **changes** ReplaySubject emits all Module definitions loaded throughout the lifespan of this class.

**Source**

packages/core/src/kernel/kernel.ts:170

**core.Kernel.****imports***(Private Readonly Property)*

Internal Mapping to keep track of all via importmaps declared Module identifiers to their corresponding paths. This map is used for house-keeping, e.g., to prevent the same Module identifier to be defined multiple times.

**Source**

packages/core/src/kernel/kernel.ts:178

**core.Kernel.****loaders***(Private Readonly Property)*

Internal Mapping of all Modules **loaders** to a ReplaySubject. This ReplaySubject tracks the loading process as such, that it emits the Module definition once the respective Module is fully loaded (including dependencies etc.) and then completes.

**Source**

packages/core/src/kernel/kernel.ts:187

**core.Kernel.****shimmed***(Private Readonly Property)*

Internally used string to suffix the `importmap` and `module` types of `HTMLScriptElements` with, if applicable. This string is set to whatever trails the type of `HTMLScriptElements` encountered upon initialization, iff their type starts with `importmap`.

**Source**`packages/core/src/kernel/kernel.ts:195`**core.****Kernel***(Namespace)*

The **Kernel** namespace contains types and interfaces used and intended to be used in conjunction with the Singleton Kernel class.

**See**

Kernel

**Source**`packages/core/src/kernel/kernel.ts:16, packages/core/src/kernel/kernel.ts:159`**core.Kernel.****Digest***(Type alias)*

String literal helper type. Enforces any assigned string to represent a browser-parsable **Digest** hash. A **Digest** hash is used to represent a hash for Subresource Integrity validation.

**Example**

A valid **Digest**:

```
import { type Kernel } from '@sgrud/core';

const digest: Kernel.Digest = 'sha256-[...]';
```

**Source**`packages/core/src/kernel/kernel.ts:31`**core.Kernel.****Module***(Interface)*

Interface describing the shape of a **Module** while being aligned with well-known `package.json` fields. This interface additionally specifies optional `sgrudDependencies` and `webDependencies` mappings, which both are used by the Kernel to determine SGRUD module dependencies and runtime dependencies.

**Example**

An exemplary **Module** definition:

```
import { type Kernel } from '@sgrud/core';

const module: Kernel.Module = {
  name: 'module',
  version: '0.0.0',
  exports: './module.exports.js',
  unpkg: './module.unpkg.js',
  sgrudDependencies: {
    sgrudDependency: '^0.0.1'
  },
  webDependencies: {
    webDependency: {
      exports: {
        webDependency: './webDependency.exports.js'
      },
      unpkg: [
```

```
        './webDependency.unpkg.js'  
      ]  
    }  
  }  
};
```

**Source**

packages/core/src/kernel/kernel.ts:66

**core.Kernel.Module.****digest**

*(Optional Readonly Property)*

Optional bundle Digests. If hashes are supplied, they will be used to verify the Subresource Integrity of the respective bundles.

**Source**

packages/core/src/kernel/kernel.ts:94

**core.Kernel.Module.****exports**

*(Optional Readonly Property)*

Optional ESM entry point.

**Source**

packages/core/src/kernel/kernel.ts:82

**core.Kernel.Module.****name**

*(Readonly Property)*

The **name** of the Module.

**Source**

packages/core/src/kernel/kernel.ts:71

**core.Kernel.Module.****sgrudDependencies**

*(Optional Readonly Property)*

Optional SGRUD Module dependencies.

**Source**

packages/core/src/kernel/kernel.ts:99

**core.Kernel.Module.****unpkg**

*(Optional Readonly Property)*

Optional UMD entry point.

**Source**

packages/core/src/kernel/kernel.ts:87

**core.Kernel.Module.****version**

*(Readonly Property)*

The Module version, formatted as according to the semver specifications.

**Source**

packages/core/src/kernel/kernel.ts:77

**core.Kernel.Module.****webDependencies**

*(Optional Readonly Property)*

Optional WebDependency mapping.

**Source**

packages/core/src/kernel/kernel.ts:104

**core.Kernel.****WebDependency**

*(Interface)*

Interface describing runtime dependencies of a Module. A Module may specify an array of UMD bundles to be loaded by the Kernel through the `unpkg` property. A Module may also specify a mapping of `import` specifiers to Module-relative paths through the `exports` property. Every specified **WebDependency** is loaded before respective bundles of the Module, which depends on the specified **WebDependency**, will be loaded themselves.

**Example**

An exemplary **webDependency** definition:

```
import { type Kernel } from '@sgrud/core';

const webDependency: Kernel.WebDependency = {
  exports: {
    webDependency: './webDependency.exports.js'
  },
  unpkg: [
    './webDependency.unpkg.js'
  ]
};
```

**Source**

packages/core/src/kernel/kernel.ts:132

**core.Kernel.WebDependency.****exports**

*(Optional Readonly Property)*

Optional ESM runtime dependencies.

**Source**

packages/core/src/kernel/kernel.ts:137

**core.Kernel.WebDependency.****unpkg**

*(Optional Readonly Property)*

Optional UMD runtime dependencies.

**Source**

packages/core/src/kernel/kernel.ts:142

**core.****Linker**

*(Class)*

The Singleton **Linker** class provides the means to lookup and retrieve instances of Targeted constructors. The **Linker** is used throughout the SGRUD client libraries, e.g., by the Factor decorator, to provide and retrieve different centrally provisioned class instances.

**Decorator**

Singleton

**Example**

Preemptively link an instance:

```
import { Linker } from '@sgrud/core';
import { Service } from './service';

new Linker<typeof Service>([
  [Service, new Service('linked')]
]);
```

**Type parameters**

Name	Type	Description
K	extends () => V	The Targeted constructor type.
V	InstanceType<K>	The Targeted InstanceType.

**Hierarchy**

- Map<K, V>
- Linker

**Source**

packages/core/src/linker/linker.ts:35

**core.Linker.****constructor***(Constructor)***Signature**

new Linker&lt;K, V&gt;(entries?)

**Type parameters**

Name	Type
K	extends () => V
V	InstanceType<K>

**Parameters**

Name	Type
entries?	null   readonly readonly [K, V][]

**Signature**

new Linker&lt;K, V&gt;(iterable?)

**Type parameters**

Name	Type
K	extends () => V
V	InstanceType<K>

**Parameters**

Name	Type
iterable?	null   Iterable<readonly [K, V]>

**Source**

node\_modules/typescript/lib/lib.es2015.collection.d.ts:50, node\_modules/typescript/lib/lib.es2015.collection.d.ts:51

**core.Linker.****get***(Method)*

Overridden **get** method. Calling this method looks up the linked instance based on the supplied target constructor. If no linked instance is found, one is created by calling the new operator on the target constructor. Therefor the target constructors must not require parameters.

**Example**

Retrieve a linked instance:

```
import { Linker } from '@sgrud/core';
import { Service } from './service';

new Linker<typeof Service>().get(Service);
```

**Signature**

get(target): V

**Returns**

The already linked or a newly constructed and linked instance.

**Parameters**

Name	Type	Description
target	K	The target constructor for which to retrieve an instance.

**Source**

packages/core/src/linker/linker.ts:58

**core.Linker.****getAll***(Method)*

The **getAll** method returns all linked instances, which satisfy instanceof target. Use this method when multiple linked target constructors extend the same base class and are to be retrieved.

**Example**

Retrieve all linked instances of a Service:

```
import { Linker } from '@sgrud/core';
import { Service } from './service';

new Linker<typeof Service>().getAll(Service);
```

**Signature**

getAll(target): V[]

**Returns**

All already linked instance of the target constructor.

**Parameters**

Name	Type	Description
target	K	The target constructor for which to retrieve instances.

**Source**

packages/core/src/linker/linker.ts:85

**core.**

## Merge

*(Type alias)*

Type helper to convert union types (A | B) to intersection types (A & B).

**Remarks**

<https://github.com/microsoft/TypeScript/issues/29594>

**Type parameters**

Name	Description
T	The union type to <b>Merge</b> .

**Source**

packages/core/src/typing/merge.ts:8

**core.**

## Mutable

*(Type alias)*

Type helper marking the supplied type as **Mutable** (opposed to readonly).

**Remarks**

<https://github.com/Microsoft/TypeScript/issues/24509>

**Type parameters**

Name	Type	Description
T	extends object	The readonly type to make <b>Mutable</b> .

**Source**

packages/core/src/typing/mutable.ts:8

**core.**

## Provide

*(Type alias)*

Type helper enforcing the provide symbol property to contain a magic string (typed as Registration) on base constructors decorated with the corresponding Provide decorator. The **Provide** type helper is also used by the Provider decorator.

**See**

Provide

**Type parameters**

Name	Type	Description
K	extends Registration	The magic string Registration type.
V	extends (...args: any[]) => InstanceType<V>	The registered class constructor type.

**Source**

packages/core/src/super/provide.ts:61, packages/core/src/super/provide.ts:19

**core.**

## Provide

*(Function)*

Class decorator factory. **Provides** the decorated class to extending classes. Applying the **Provide** decorator enforces the Provide type which entails the declaration of a static provide property typed as Registration. The magic string assigned to this static property is used by the Provider factory function to get base classes from the Registry.

**Example**

**Provide** a base class:

```
import { Provide, provide } from '@sgrud/core';

@Provide()
export abstract class Base {

  public static readonly [provide] = 'sgrud.example.Base' as const;

}
```

**See**

Provider, Registry

**Signature**

Provide<V, K>(): (constructor: V) => void

**Returns**

A class constructor decorator.

**Type parameters**

Name	Type	Description
V	extends Provide<K, V>	The registered class constructor type.
K	extends Registration = V[typeof provide]	The magic string Registration type.

**Source**

packages/core/src/super/provide.ts:61, packages/core/src/super/provide.ts:19

**core.**

## Provider

(Function)

**Provider** of base classes. Extending this mixin-style function while supplying the `typeof` a `Provided` constructor enforces type safety and hinting on the supplied magic string and the resulting class which extends this **Provider** mixin. The main purpose of this pattern is bridging module gaps by de-coupling bundle files while maintaining a well-defined prototype chain. This still requires the base class to be defined (and `Provided`) before extension but allows intellisense'd OOP patterns across multiple modules while maintaining runtime language specifications.

**Example**

Extend a provided class:

```
import { Provider } from '@sgrud/core';
import { type Base } from 'example-module';

export class Class
  extends Provider<typeof Base>('sgrud.example.Base') {

  public constructor(...args: any[]) {
    super(...args);
  }

}
```

**See**

Provide, Registry

**Signature**

Provider<V, K>(provider): V

**Returns**

The constructor which Provides the Registration.



**Type parameters**

Name	Type	Description
V	extends Provide<K, V>	The registered class constructor type.
K	extends Registration = V[typeof provide]	The magic string Registration type.

**Parameters**

Name	Type	Description
provider	K	A magic string to retrieve the provider by.

**Source**

packages/core/src/super/provider.ts:64, packages/core/src/super/provider.ts:13

**core.****Provider**

(Interface)

Type helper to allow referencing Provided constructors as new-able targets. Used and intended to be used in conjunction with the Provider decorator.

**See**

Provider

**Type parameters**

Name	Description
V	Instance type of the registered class constructor.

**Source**

packages/core/src/super/provider.ts:64, packages/core/src/super/provider.ts:13

**core.Provider.****[provide]**

(Readonly Property)

Enforced provider contract.

**Source**

packages/core/src/super/provider.ts:18

**core.Provider.****constructor**

(Constructor)

Enforced constructor contract.

**Signature**

new Provider(...args)

**Parameters**

Name	Type	Description
...args	any[]	The default class constructor rest parameter.

**Source**

packages/core/src/super/provider.ts:64, packages/core/src/super/provider.ts:13

**core.**

## Proxy

*(Abstract Class)*

Abstract **Proxy** base class to implement Request interceptors, on the client side. By extending this abstract base class and providing the extending class to the Linker, e.g., by Targeting it, the class's handle method will be called with the Request details (which could have been modified by a previous **Proxy**) and the next Handler, whenever a request is fired through the Http class.

### Decorator

Provide

### Example

Simple **Proxy** intercepting file: requests:

```
import { type Http, Provider, type Proxy, Target } from '@sgrud/core';
import { type Observable, of } from 'rxjs';
import { file } from './file';
```

```
@Target()
export class FileProxy
  extends Provider<typeof Proxy>('sgrud.core.Proxy') {

  public override handle(
    request: Http.Request,
    handler: Http.Handler
  ): Observable<Http.Response> {
    if (request.url.startsWith('file:')) {
      return of<Http.Response>(file);
    }

    return handler.handle(request);
  }
}
```

See

Http

### Hierarchy

- Proxy
  - Transit

### Source

packages/core/src/http/proxy.ts:44

**core.Proxy.**

## [provide]

*(Static Readonly Property)*

Magic string by which this class is provided.

See

provide

### Source

packages/core/src/http/proxy.ts:51

**core.Proxy.**

## handle

*(Abstract Method)*

The **handle** method of linked classes extending the Proxy base class is called whenever an Request is fired. The extending class can either pass the request to the next handler, with or without modifying it, or an interceptor can chose to completely handle a request by itself through returning an Observable Response.

### Signature

handle(request, handler): Observable&lt;Response&lt;any&gt;&gt;

**Returns**

An Observable of the **handled** Response.

**Parameters**

Name	Type	Description
request	Request	The Request to be <b>handled</b> .
handler	Handler	The next Handler to <b>handle</b> the request.

**Source**

packages/core/src/http/proxy.ts:64

**core.**

## Registration

(Type alias)

String literal helper type. Enforces any assigned string to contain at least three dots. **Registrations** are used by the Registry to alias classes extending the base Provider as magic strings and should represent sane module paths in dot-notation.

**Example**

Library-wide **Registration** pattern:

```
import { type Registration } from '@sgrud/core';

const registration: Registration = 'sgrud.module.ClassName';
```

**See**

Registry

**Source**

packages/core/src/super/registry.ts:20

**core.**

## Registry

(Class)

The Singleton **Registry** is a mapping used by the Provider to lookup Provided constructors by Registrations upon class extension. Magic strings should represent sane module paths in dot-notation. Whenever a currently not registered constructor is requested, an intermediary class is created, cached internally and returned. When the actual constructor is registered later, the previously created intermediary class is removed from the internal caching and further steps are taken to guarantee the transparent addressing of the actual constructor through the dropped intermediary class.

**Decorator**

Singleton

**See**

Provide, Provider

**Type parameters**

Name	Type	Description
K	extends Registration	The magic string Registration type.
V	extends (...args: any[]) => InstanceType<V>	The registered class constructor type.

**Hierarchy**

- Map<K, V>
  - **Registry**

**Source**

packages/core/src/super/registry.ts:49

**core.Registry.****constructor***(Constructor)*

Public **constructor**. The constructor of this class accepts the same parameters as its overridden super **Map constructor** and acts the same. I.e., through instantiating this Singleton class and passing a list of tuples of Registrations and their corresponding class constructors, these tuples may be preemptively registered.

**Example**

Preemptively provide a class constructor by magic string:

```
import { type Registration, Registry } from '@sgrud/core';
import { Service } from './service';

const registration = 'sgrud.example.Service';
new Registry<Registration, typeof Service>([
  [registration, Service]
]);
```

**Signature**

new Registry<K, V>(tuples?)

**Type parameters**

Name	Type
K	extends Registration
V	extends (...args: any[]) => InstanceType<V>

**Parameters**

Name	Type	Description
tuples?	Iterable<[K, V]>	An Iterable of tuples provide.

**Source**

packages/core/src/super/registry.ts:94

**core.Registry.****get***(Method)*

Overridden **get** method. Looks up the Provided constructor by magic string. If no provided constructor is found, an intermediary class is created, cached internally and returned. While this intermediary class and the functionality supporting it take care of inheritance, i.e., allow forward-referenced base classes to be extended, it cannot substitute for the actual extended constructor. Therefore, the static extension of forward-referenced classes is possible, but as long as the actual extended constructor is not registered (and therefore the intermediary class still caches the inheritance chain), the extending classes cannot be instantiated, called etc. Doing so will result in a ReferenceError being thrown.

**Throws**

A ReferenceError when a cached class is invoked.

**Example**

Retrieve a provided constructor by magic string:

```
import { type Registration, Registry } from '@sgrud/core';
import { type Service } from 'example-module';

const registration = 'sgrud.example.Service';
new Registry<Registration, typeof Service>().get(registration);
```

**Signature**

get(registration): V

**Returns**

The Provided constructor or a cached intermediary.

**Parameters**

Name	Type	Description
registration	K	The magic string to <b>get</b> the class constructor by.

**Source**

packages/core/src/super/registry.ts:134

**core.Registry.****set**

*(Method)*

Overridden **set** method. Whenever a class constructor is provided by magic string through calling this method, a test is run, whether this constructor was previously requested and therefore cached as intermediary class. If so, the intermediary class is removed from the internal mapping and further steps are taken to guarantee the transparent addressing of the newly provided constructor through the previously cached and now dropped intermediary class.

**Example**

Preemptively provide a constructor by magic string:

```
import { type Registration, Registry } from '@sgrud/core';
import { Service } from './service';

const registration = 'sgrud.example.Service';
new Registry<Registration, typeof Service>().set(registration, Service);
```

**Signature**

set(registration, constructor): Registry<K, V>

**Returns**

This Registry instance.

**Parameters**

Name	Type	Description
registration	K	The magic string to <b>set</b> the class constructor by.
constructor	V	The constructor to register for the registration.

**Source**

packages/core/src/super/registry.ts:186

**core.Registry.****cached**

*(Private Readonly Property)*

Internal Mapping of all **cached**, i.e., forward-referenced, class constructors. Whenever a constructor, which is not currently registered, is requested as a Provider, an intermediary class is created and stored within this map until the actual constructor is registered. As soon as this happens, the intermediary class is removed from this map and further steps are taken to guarantee the transparent addressing of the actual constructor through the dropped intermediary class.

**Source**

packages/core/src/super/registry.ts:63

**core.Registry.****caches**

*(Private Readonly Property)*

Internally used WeakSet containing all intermediary classes created upon requesting a currently not registered constructor as provider. This set is used internally to check if a intermediary class has already been replaced by the actual constructor.

**Source**

packages/core/src/super/registry.ts:71

core.

**Singleton**

(Function)

Class decorator factory. Enforces a transparent **Singleton** pattern on the decorated class. When calling the new operator on a decorated class for the first time, an instance of the decorated class is created using the supplied arguments, if any. This instance will remain the **Singleton** instance of the decorated class indefinitely. When calling the new operator on a decorated class already instantiated, the **Singleton** pattern is enforced and the previously constructed instance is returned. Instead, if provided, the apply callback is fired with the **Singleton** instance and the new invocation parameters.

**Example**

**Singleton** class:

```
import { Singleton } from '@sgrud/core';
```

```
@Singleton()
```

```
export class Service {}
```

```
new Service() === new Service(); // true
```

**Signature**

Singleton<T>(apply?): (constructor: T) => T

**Returns**

A class constructor decorator.

**Type parameters**

Name	Type	Description
T	extends (...args: any[]) => any	The type of the decorated constructor.

**Parameters**

Name	Type	Description
apply?	(self: InstanceType<T>, args: ConstructorParameters<T>) => InstanceType<T>	The callback to apply on subsequent new invocations.

**Source**

packages/core/src/utility/singleton.ts:27

core.

**Spawn**

(Function)

This prototype property decorator factory **Spawns** a Worker and wraps and assigns the resulting Remote to the decorated prototype property.

**Example**

**Spawn** a Worker:

```
import { Spawn, type Thread } from '@sgrud/core';
import { type ExampleWorker } from 'example-worker';
```

```
export class ExampleWorkerHandler {
```

```
  @Spawn('example-worker')
```

```
  public readonly worker!: Thread<ExampleWorker>;
```

```
}
```

**See**

Thread

**Signature**

```
Spawn(worker, source?): (prototype: object, propertyKey: PropertyKey) => void
```

**Returns**

A prototype property decorator.

**Parameters**

Name	Type	Description
worker	string   Endpoint   NodeEndpoint	The worker module name or Endpoint to <b>Spawn</b> .
source?	string	An optional Module source.

**Source**

packages/core/src/thread/spawn.ts:32

**core.****Symbol***(Function)*

Proxy around the built-in Symbol object, returning the requested symbol or the name of the requested symbol prefixed with '@@'.

**Signature**

```
Symbol(description?): symbol
```

**Parameters**

Name	Type
description?	string   number

**Source**

packages/core/src/utility/symbols.ts:5

**core.****Target***(Type alias)*Type helper to allow Factoring **Targeted** constructors with required arguments. Used and to be used in conjunction with the Target decorator.**Signature**

```
(...args)
```

**Parameters**

Name	Type
...args	any[]

**Type parameters**

Name	Description
V	The <b>Targeted</b> InstanceType.

**Source**

packages/core/src/linker/target.ts:56, packages/core/src/linker/target.ts:10

**core.**

## Target

*(Function)*

Class decorator factory. Links the **Targeted** constructor to its corresponding instance by applying the supplied `factoryArgs`. Employ this helper to link **Targeted** constructors with required arguments. Supplying a `target` constructor overrides its linked instance, if any, with the constructed instance.

### Example

**Target** a service:

```
import { Target } from '@sgrud/core';
```

```
@Target(['default'])
export class Service {

  public constructor(
    public readonly param: string
  ) {}
}
```

### Example

Factor a **Targeted** service:

```
import { Factor, type Target } from '@sgrud/core';
import { Service } from './service';

export class ServiceHandler {

  @Factor<Target<Service>>(() => Service)
  public readonly service!: Service;

}
```

### See

Factor, Linker

### Signature

`Target<K>(factoryArgs?, target?): (constructor: K) => void`

### Returns

A class constructor decorator.

### Type parameters

Name	Type	Description
K	extends (...args: any[]) => any	The <b>Targeted</b> constructor type.

### Parameters

Name	Type	Description
factoryArgs?	ConstructorParameters<K>	The arguments for the <b>Targeted</b> constructor.
target?	K	An optional <b>Target</b> constructor to override.

### Source

packages/core/src/linker/target.ts:56, packages/core/src/linker/target.ts:10

**core.**

## Thread

*(Type alias)*

Type alias describing an exposed class in a remote context. Represented by wrapping a `Remote` in a `Promise`. Used and intended to be used in conjunction with the `Thread` decorator.



**See**

Thread

**Type parameters**

Name	Description
T	The Remote <b>Thread</b> type.

**Source**

packages/core/src/thread/thread.ts:32, packages/core/src/thread/thread.ts:13

**core.****Thread***(Function)*Class decorator factory. exposes an instance of the decorated class as Worker **Thread**.**Example**ExampleWorker **Thread**:

```
import { Thread } from '@sgrud/core';
```

```
@Thread()
export class ExampleWorker {}
```

**See**

Spawn

**Signature**

Thread(): (constructor: () =&gt; any) =&gt; void

**Returns**

A class constructor decorator.

**Source**

packages/core/src/thread/thread.ts:32, packages/core/src/thread/thread.ts:13

**core.****Transit***(Class)*

The Targeted Singleton **Transit** class is a built-in Proxy intercepting all connections opened by the Http class. This Proxy implements the `Symbol.observable` pattern, through which it emits an array of all currently open Requests every time a new Request is fired or a previously fired Request completes.

**Decorator**

Target, Singleton

**See**

Http, Proxy

**Hierarchy**

- Proxy<this>
  - **Transit**

**Source**

packages/core/src/http/transit.ts:26

**core.Transit.****[provide]***(Static Readonly Property)*

Magic string by which this class is provided.

**See**

provide

**Source**

packages/core/src/http/proxy.ts:51

**core.Transit.****[observable]***(Method)*

Well-known `Symbol.observable` method returning a `Subscribable`. The returned `Subscribable` emits all active `Requests` in an array, whenever this list changes. Using the returned `Subscribable`, e.g., a load indicator can easily be implemented.

**Example**

Subscribe to the currently active `Request`:

```
import { Transit, Linker } from '@sgrud/core';
import { from } from 'rxjs';

const transit = new Linker<typeof Transit>().get(Transit);
from(transit).subscribe(console.log);
```

**Signature**

`[observable](): Subscribable<Response<any>[]>`

**Returns**

A `Subscribable` emitting all active `Request`.

**Source**

packages/core/src/http/transit.ts:70

**core.Transit.****constructor***(Constructor)*

Public **constructor**. Called by the `Target` decorator to link this `Proxy` so it may be used by the `Http` class.

**Signature**

`new Transit()`

**Source**

packages/core/src/http/transit.ts:45

**core.Transit.****handle***(Method)*

Overridden **handle** method of the `Proxy` base class. Mutates the `request` to also emit progress events while it is running. These progress events will be consumed by the `Transit` interceptor and re-supplied via the `Subscribable` returned by the `Symbol.observable` method.

**Signature**

`handle(request, handler): Observable<Response<any>>`

**Returns**

An `Observable` of the **handled** `Response`.

**Parameters**

Name	Type	Description
request handler	Request Handler	The Request to be <b>handled</b> . The next Handler to <b>handle</b> the request.

**Source**

packages/core/src/http/transit.ts:84

**core.Transit.****changes**

*(Private Readonly Property)*

The **changes** Subject emits every time a request is added to or deleted from the internal requests mapping.

**Source**

packages/core/src/http/transit.ts:33

**core.Transit.****requests**

*(Private Readonly Property)*

Internal Mapping of all running requests. Mutating this map should be accompanied by an emittance of the changes Subject.

**Source**

packages/core/src/http/transit.ts:39

**core.****TypeOf**

*(Abstract Class)*

Strict type-assertion and runtime type-checking utility. When type-checking variables in the global scope, e.g., window or process, make use of the globalThis object.

**Example**

Type-check global context:

```
import { TypeOf } from '@sgrud/core';

TypeOf.process(globalThis.process); // true if running in node context
TypeOf.window(globalThis.window); // true if running in browser context
```

**Source**

packages/core/src/utility/type-of.ts:15

**core.TypeOf.****array**

*(Static Method)*

Type-check value for unknown[].

**Example**

Type-check null for unknown[]:

```
import { TypeOf } from '@sgrud/core';

TypeOf.array(null); // false
```

**Signature**

array(value): value is unknown[]

**Returns**

Whether value is of type `unknown[]`.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:31

**core.TypeOf.****boolean**

*(Static Method)*

Type-check value for boolean.

**Example**

Type-check null for boolean:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.boolean(null); // false
```

**Signature**

`boolean(value): value is boolean`

**Returns**

Whether value is of type boolean.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:49

**core.TypeOf.****date**

*(Static Method)*

Type-check value for Date.

**Example**

Type-check null for Date:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.date(null); // false
```

**Signature**

`date(value): value is Date`

**Returns**

Whether value is of type Date.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:67

**core.TypeOf.****function**

*(Static Method)*

Type-check value for Function.

**Example**

Type-check null for Function:

```
import { TypeOf } from '@sgrud/core';  
  
TypeOf.function(null); // false
```

**Signature**

function(value): value is Function

**Returns**

Whether value is of type Function.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:85

**core.TypeOf.****null**

*(Static Method)*

Type-check value for null.

**Example**

Type-check null for null:

```
import { TypeOf } from '@sgrud/core';  
  
TypeOf.null(null); // true
```

**Signature**

null(value): value is null

**Returns**

Whether value is of type null.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:103

**core.TypeOf.****number***(Static Method)*

Type-check value for number.

**Example**

Type-check null for number:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.number(null); // false
```

**Signature**

number(value): value is number

**Returns**

Whether value is of type number.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:121

**core.TypeOf.****object***(Static Method)*

Type-check value for object.

**Example**

Type-check null for object:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.object(null); // false
```

**Signature**

object(value): value is object

**Returns**

Whether value is of type object.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:139

**core.TypeOf.****process***(Static Method)*

Type-check value for NodeJS.Process.

**Example**

Type-check null for NodeJS.Process:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.process(null); // false
```

#### Signature

process(value): value is Process

#### Returns

Whether value is of type NodeJS.Process.

#### Parameters

Name	Type	Description
value	unknown	The value to type-check.

#### Source

packages/core/src/utility/type-of.ts:157

#### core.TypeOf.

### promise

(Static Method)

Type-check value for Promise<unknown>.

#### Example

Type-check null for Promise<unknown>:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.promise(null); // false
```

#### Signature

promise(value): value is Promise<unknown>

#### Returns

Whether value is of type Promise<unknown>.

#### Parameters

Name	Type	Description
value	unknown	The value to type-check.

#### Source

packages/core/src/utility/type-of.ts:175

#### core.TypeOf.

### regex

(Static Method)

Type-check value for RegExp.

#### Example

Type-check null for RegExp:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.regex(null); // false
```

#### Signature

regex(value): value is RegExp

#### Returns

Whether value is of type RegExp.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:193

**core.TypeOf.****string**

*(Static Method)*

Type-check value for string.

**Example**

Type-check null for string:

```
import { TypeOf } from '@sgrud/core';  
  
TypeOf.string(null); // false
```

**Signature**

string(value): value is string

**Returns**

Whether value is of type string.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:211

**core.TypeOf.****undefined**

*(Static Method)*

Type-check value for undefined.

**Example**

Type-check null for undefined:

```
import { TypeOf } from '@sgrud/core';  
  
TypeOf.undefined(null); // false
```

**Signature**

undefined(value): value is undefined

**Returns**

Whether value is of type undefined.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:229



**core.TypeOf.****url***(Static Method)*

Type-check value for URL.

**Example**

Type-check null for URL:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.url(null); // false
```

**Signature**

url(value): value is URL

**Returns**

Whether value is of type URL.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:247

**core.TypeOf.****window***(Static Method)*

Type-check value for Window.

**Example**

Type-check null for Window:

```
import { TypeOf } from '@sgrud/core';
```

```
TypeOf.window(null); // false
```

**Signature**

window(value): value is Window

**Returns**

Whether value is of type Window.

**Parameters**

Name	Type	Description
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:265

**core.TypeOf.****test***(Static Private Method)*

Type-check value for type.

**Signature**

test(type, value): boolean

**Returns**

Whether value is type.

**Parameters**

Name	Type	Description
type	string	The type to check for.
value	unknown	The value to type-check.

**Source**

packages/core/src/utility/type-of.ts:276

**core.TypeOf.****constructor**

(Private Constructor)

Private **constructor** (which should never be called).

**Throws**

A TypeError upon construction.

**Signature**

```
new TypeOf()
```

**Source**

packages/core/src/utility/type-of.ts:285

**core.****assign**

(Function)

**assigns** (deep copies) the values of all of the enumerable own properties from one or more sources to a target. The last value within the last sources object takes precedence over any previously encountered values.

**Example**

**assign** nested properties:

```
import { assign } from '@sgrud/core';

assign(
  { one: { one: true }, two: false },
  { one: { key: null } },
  { two: true }
);

// { one: { one: true, key: null }, two: true }
```

**Signature**

```
assign<T, S>(target, ...sources): T & Merge<S[number]>
```

**Returns**

The **assigned**-to target object.

**Type parameters**

Name	Type	Description
T	extends Record<PropertyKey, any>	The type of the target object.
S	extends Record<PropertyKey, any>[]	The types of the sources objects.

**Parameters**

Name	Type	Description
target ...sources	T [...S[]]	The target object to <b>assign</b> properties to. An array of sources from which to deep copy properties.

**Source**

packages/core/src/utility/assign.ts:29

**core.**

**pluralize**

(Function)

**pluralize** words of the English language.

**Example**

**Pluralize** 'money':

```
import { pluralize } from '@sgrud/core';

pluralize('money'); // 'money'
```

**Example**

**Pluralize** 'thesis':

```
import { pluralize } from '@sgrud/core';

pluralize('thesis'); // 'theses'
```

**Signature**

pluralize(singular): string

**Returns**

The **pluralized** form of singular.

**Parameters**

Name	Type	Description
singular	string	An English word in singular form.

**Source**

packages/core/src/utility/pluralize.ts:23

**core.**

**provide**

(Const Variable)

Unique symbol used as property key by the Provide type constraint.

**Source**

packages/core/src/super/provide.ts:6

**core.**

**semver**

(Function)

Best-effort **semver** matcher. The supplied version will be tested against the supplied range.

**Example**

Test '1.2.3' against '>2 <1 || ~1.2.\*':

```
import { semver } from '@sgrud/core';

semver('1.2.3', '>2 <1 || ~1.2.*'); // true
```

**Example**

Test '1.2.3' against '~1.1':

```
import { semver } from '@sgrud/core';

semver('1.2.3', '~1.1'); // false
```

**Signature**

semver(version, range): boolean

**Returns**

Whether version satisfies range.

**Parameters**

Name	Type	Description
version	string	The to-be tested semantic version string.
range	string	The range to test the version against.

**Source**

packages/core/src/kernel/semver.ts:25

## @sgrud/data Module

**@sgrud/data** - The SGRUD Data Model.

The functions and classes found within the **@sgrud/data** module are intended to ease the type safe data handling, i.e., retrieval, mutation and storage, within applications built upon the SGRUD client libraries. By extending the Model class and applying adequate decorators to the contained properties, the resulting extension will, in its static context, provide all necessary means to interact directly with the underlying repository, while the instance context of any class extending the abstract Model base class will inherit methods to observe changes to its instance field values, selectively complement the instance with fields from the backing data storage via type safe graph representations and to delete the respective instance from the data storage.

**Source**

packages/data/index.ts:1

**data.**

## Enum

(Abstract Class)

Abstract **Enum** helper class. This class is used by the Model to detect **Enumerations** within a Graph, as **Enumerations** (in contrast to plain strings) must not be quoted. This class should never be instantiated manually, but instead is used internally by the enumerate function.

**See**

enumerate

**Hierarchy**

- String
  - Enum

**Source**

packages/data/src/model/enum.ts:10

**data.Enum.**

## constructor

(Private Constructor)

Private **constructor** (which should never be called).

**Throws**

A `TypeError` upon construction.

**Signature**

```
new Enum()
```

**Source**

packages/data/src/model/enum.ts:18

**data.**

## HasMany

(Function)

Model field decorator factory. Using this decorator, Models can be enriched with one-to-many associations to other Models. The value for the `typeFactory` argument has to be another Model. By applying this decorator, the decorated field will (depending on the `transient` argument value) be taken into account when serializing or treemapping the Model containing the decorated field.

**Example**

Model with a one-to-many association:

```
import { HasMany, Model } from '@sgrud/data';
import { OwnedModel } from '../owned-model';

export class ExampleModel extends Model<ExampleModel> {

  @HasMany(() => OwnedModel)
  public field?: OwnedModel[];

  protected [Symbol.toStringTag]: string = 'ExampleModel';
}
```

**See**

Model, HasOne, Property

**Signature**

```
HasMany<T>(typeFactory, transient?): <M>(model: M, field: Field<M>) => void
```

**Returns**

A Model field decorator.

**Type parameters**

Name	Type	Description
T	extends <code>Type&lt;Model&lt;any&gt;, T&gt;</code>	The field value constructor type.

**Parameters**

Name	Type	Default value	Description
<code>typeFactory</code>	<code>() =&gt; T</code>	undefined	A forward reference to the field value constructor.
<code>transient</code>	boolean	false	Whether the decorated field is transient.

**Source**

packages/data/src/relation/has-many.ts:46

**data.**

## HasOne

(Function)

Model field decorator factory. Using this decorator, Models can be enriched with one-to-one associations to other Models. The value for the

`typeFactory` argument has to be another `Model`. By applying this decorator, the decorated field will (depending on the `transient` argument value) be taken into account when serializing or treemapping the `Model` containing the decorated field.

#### Example

Model with a one-to-one association:

```
import { HasOne, Model } from '@sgrud/data';
import { OwnedModel } from './owned-model';

export class ExampleModel extends Model<ExampleModel> {

  @HasOne(() => OwnedModel)
  public field?: OwnedModel;

  protected [Symbol.toStringTag]: string = 'ExampleModel';
}
```

#### See

Model, HasMany, Property

#### Signature

`HasOne<T>(typeFactory, transient?): <M>(model: M, field: Field<M>) => void`

#### Returns

A Model field decorator.

#### Type parameters

Name	Type	Description
T	extends <code>Type&lt;Model&lt;any&gt;, T&gt;</code>	The field value constructor type.

#### Parameters

Name	Type	Default value	Description
<code>typeFactory</code>	<code>() =&gt; T</code>	undefined	A forward reference to the field value constructor.
<code>transient</code>	boolean	false	Whether the decorated field is transient.

#### Source

packages/data/src/relation/has-one.ts:46

data.

## HttpQuerier

(Class)

The **HttpQuerier** class implements an Http based Querier, i.e., an extension of the abstract Querier base class, allowing for Model queries to be executed via HTTP. To use this class, provide it to the Linker by either extending it, and decorating the extending class with the Target decorator, or by preemptively supplying an instance of this class to the Linker.

#### Example

Provide the **HttpQuerier** to the Linker:

```
import { Linker } from '@sgrud/core';
import { HttpQuerier } from '@sgrud/data';

new Linker<typeof HttpQuerier>([
  [HttpQuerier, new HttpQuerier('https://api.example.com')]
]);
```

#### See

Model, Querier

**Hierarchy**

- Querier<this>
  - **HttpQuerier**

**Source**

packages/data/src/querier/http.ts:28

**data.HttpQuerier.****[provide]**

*(Static Readonly Property)*

Magic string by which this class is provided.

**See**

provide

**Source**

packages/data/src/querier/querier.ts:96

**data.HttpQuerier.****commit**

*(Method)*

Overridden **commit** method of the Querier base class. When this Querier is made available via the Linker, this overridden **commit** method is called when this Querier claims the highest priority to **commit** an Operation, depending on the Model from which the Operation originates.

**Throws**

An Observable of an AggregateError.

**Signature**

commit(operation, variables?): Observable<unknown>

**Returns**

An Observable of the committed Operation.

**Parameters**

Name	Type	Description
operation	Operation	The Operation to be <b>committed</b> .
variables?	Variables	Any Variables within the operation.

**Source**

packages/data/src/querier/http.ts:82

**data.HttpQuerier.****constructor**

*(Constructor)*

Public **constructor** consuming the HTTP endpoint Model queries should be committed against, and an dynamic or static *prioritize* value. The *prioritize* value may either be a mapping of Models to corresponding priorities or a static priority for this Querier.

**Signature**

new HttpQuerier(endpoint, prioritize?)

**Parameters**

Name	Type	Default value	Description
endpoint	string	undefined	The HTTP endpoint to commit queries against.

prioritize	number   Map<Type<Model<any>>, number>	0	The dynamic or static prioritization.
------------	--	---	--

**Source**

packages/data/src/querier/http.ts:50

**data.HttpQuerier.**

**priority**

*(Method)*

Overridden **priority** method of the Querier base class. When an Operation is to be committed, this method is called with the respective model Type and returns the claimed **priority** to commit this Model.

**Signature**

priority(model): number

**Returns**

The numeric **priority** of this Querier implementation.

**Parameters**

Name	Type	Description
model	Type<Model<any>>	The Model to be committed.

**Source**

packages/data/src/querier/http.ts:108

**data.HttpQuerier.**

**types**

*(Readonly Property)*

A set containing the Types this Querier can handle. As HTTP connections are short-lived, the HttpQuerier may only handle one-off **types**, namely 'mutation' and 'query'.

**Source**

packages/data/src/querier/http.ts:36

**data.HttpQuerier.**

**endpoint**

*(Private Readonly Property)*

The HTTP endpoint to commit queries against.

**Source**

packages/data/src/querier/http.ts:55

**data.HttpQuerier.**

**prioritize**

*(Private Readonly Property)*

The dynamic or static prioritization.

**See**

priority

**Source**

packages/data/src/querier/http.ts:64



**data.**

## Model

(Abstract Class)

Abstract base class to implement data **Models**. By extending this abstract base class while providing the `Symbol.toStringTag` property containing the singular name of the resulting data **Model**, type safe data handling, i.e., retrieval, mutation and storage, can easily be achieved. Through the use of the static- and instance-scoped polymorphic `this`, all inherited operations warrant type safety and provide intellisense.

### Example

Extend the **Model** base class:

```
import { Model, Property } from '@sgrud/data';

export class ExampleModel extends Model<ExampleModel> {

  @Property(() => String)
  public field: string?;

  protected [Symbol.toStringTag]: string = 'ExampleModel';
}
```

### See

Querier

### Type parameters

Name	Type	Description
M	extends Model = any	The extending <b>Model</b> InstanceType.

### Source

packages/data/src/model/model.ts:18, packages/data/src/model/model.ts:126, packages/data/src/model/model.ts:323

### data.Model.

## commit

(Static Method)

Static **commit** method. Calling this method on a class extending the abstract Model base class, while supplying an operation and all its embedded variables, will dispatch the Operation to the respective Model repository through the highest priority Querier or, if no Querier is compatible, an error is thrown. This method is the entry point for all Model-related data transferral and is internally called by all other distinct methods of the Model.

### Throws

An Observable ReferenceError on incompatibility.

### Example

**commit** a query-type operation:

```
import { ExampleModel } from './example-model';

ExampleModel.commit(`query queryExample(variable: $variable) {
  result {
    field
  }
}`, {
  variable: 'value'
}).subscribe(console.log);
```

### Signature

`commit<T>(this, operation, variables?): Observable<unknown>`

### Returns

An Observable of the **committed** operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
operation	Operation	The Operation to be <b>committed</b> .
variables?	Variables	Any Variables within the operation.

**Source**

packages/data/src/model/model.ts:357

**data.Model.****deleteAll**

(Static Method)

Static **deleteAll** method. Calling this method on a class extending the Model, while supplying an array of `uuids`, will dispatch the deletion of all Model instances identified by these UUIDs to the respective Model repository by internally calling commit with suitable arguments. Through this method, bulk-deletions from the respective Model repository can be achieved.

**Example**

Drop all model instances by UUIDs:

```
import { ExampleModel } from './example-model';
```

```
ExampleModel.deleteAll([
  'b050d63f-cede-46dd-8634-a80d0563ead8',
  'a0164132-cd9b-4859-927e-ba68bc20c0ae',
  'b3fca31e-95cd-453a-93ae-969d3b120712'
]).subscribe(console.log);
```

**Signature**

deleteAll<T>(this, uuids): Observable<unknown>

**Returns**

An Observable of the deletion.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
uuids	string[]	An array of uuids of Models to be deleted.

**Source**

packages/data/src/model/model.ts:410

**data.Model.****deleteOne**

(Static Method)

Static **deleteOne** method. Calling this method on a class extending the Model, while supplying an `uuid`, will dispatch the deletion of the Model instance identified by this UUID to the respective repository by internally calling the commit operation with suitable arguments. Through this method, the deletion of a single Model instance from the respective Model repository can be achieved.

**Example**

Drop one model instance by UUID:

```
import { ExampleModel } from './example-model';

ExampleModel.deleteOne(
  '18f3aa99-afa5-40f4-90c2-71a2ecc25651'
).subscribe(console.log);
```

**Signature**

`deleteOne<T>(this, uuid): Observable<unknown>`

**Returns**

An Observable of the deletion.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
uuid	string	The uuid of the Model instance to be deleted.

**Source**

packages/data/src/model/model.ts:444

**data.Model.****findAll**

*(Static Method)*

Static **findAll** method. Calling this method on a class extending the abstract Model base class, while supplying a filter to match Model instances by and a graph containing the fields to be included in the result, will dispatch a lookup operation to the respective repository by internally calling the commit operation with suitable arguments. Through this method, the bulk-lookup of Model instances from the respective Model repository can be achieved.

**Example**

Lookup all UUIDs for model instances modified between two dates:

```
import { ExampleModel } from './example-model';

ExampleModel.findAll({
  expression: {
    conjunction: {
      operands: [
        {
          entity: {
            operator: 'GREATER_OR_EQUAL',
            path: 'modified',
            value: new Date('2021-01-01')
          }
        },
        {
          entity: {
            operator: 'LESS_OR_EQUAL',
            path: 'modified',
            value: new Date('2021-12-12')
          }
        }
      ],
      operator: 'AND'
    }
  }
})
```

```

    }
  }, [
    'uuid',
    'field'
  ]).subscribe(console.log);

```

**Signature**

`findAll<T>(this, filter, graph): Observable<Results<T>>`

**Returns**

An Observable of the find operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
filter	Filter<T>	A Filter to find Model instances by.
graph	Graph<T>	A Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:503

**data.Model.****findOne**

(Static Method)

Static **findOne** method. Calling this method on a class extending the abstract Model base class, while supplying the shape to match the Model instance by and a graph describing the fields to be included in the result, will dispatch the lookup operation to the respective repository by internally calling the commit operation with suitable arguments. Through this method, the retrieval of one specific Model instance from the respective Model repository can be achieved.

**Example**

Lookup one model instance by UUID:

```

import { ExampleModel } from './example-model';

ExampleModel.findOne({
  id: '2cfe7609-c4d9-4e4f-9a8b-ad72737db48a'
}, [
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);

```

**Signature**

`findOne<T>(this, shape, graph): Observable<T>`

**Returns**

An Observable of the find operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
shape	Shape<T>	The Shape of instance to find.
graph	Graph<T>	A Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:552

**data.Model.****saveAll**

(Static Method)

Static **saveAll** method. Calling this method on a class extending the abstract Model base class, while supplying a list of models which to save and a graph describing the fields to be returned in the result, will dispatch the save operation to the respective Model repository by internally calling the commit operation with suitable arguments. Through this method, bulk-persistence of Model instances from the respective Model repository can be achieved.

**Example**

Persist multiple Models:

```
import { ExampleModel } from './example-model';

ExampleModel.saveAll([
  new ExampleModel({ field: 'example_1' }),
  new ExampleModel({ field: 'example_2' }),
  new ExampleModel({ field: 'example_3' })
], [
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

**Signature**

saveAll<T>(this, models, graph): Observable<T[]>

**Returns**

An Observable of the save operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
models	T[]	An array of Models to be saved.
graph	Graph<T>	The Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:598

**data.Model.****saveOne**

(Static Method)

Static **saveOne** method. Calling this method on a class extending the abstract Model base class, while supplying a model which to save and a graph describing the fields to be returned in the result, will dispatch the save operation to the respective Model repository by internally calling

the commit operation with suitable arguments. Through this method, persistence of one specific Model instance from the respective Model repository can be achieved.

#### Example

Persist a model:

```
import { ExampleModel } from './example-model';

ExampleModel.saveOne(new ExampleModel({ field: 'example' })), [
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

#### Signature

saveOne<T>(this, model, graph): Observable<T>

#### Returns

An Observable of the save operation.

#### Type parameters

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

#### Parameters

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
model	T	The Model which is to be saved.
graph	Graph<T>	A Graph of fields to be returned.

#### Source

packages/data/src/model/model.ts:640

#### data.Model.

### serialize

(Static Method)

Static **serialize** method. Calling this method on a class extending the Model, while supplying a model which to **serialize** and optionally enabling shallow serialization, will return the **serialized** Shape of the Model, i.e., a plain JSON representation of all Model fields, or undefined, if the supplied model does not contain any fields or values. By serializing shallowly, only such properties defined on the supplied model are included (which means, all one-to-one and one-to-many associations are ignored). Through this method, the serialization of one specific Model instance from the respective Model repository can be achieved.

#### Example

serialize a model:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
const shape = ExampleModel.serialize(model);
console.log(shape); // { field: 'example' }
```

#### Signature

serialize<T>(this, model, shallow?): undefined | Shape<T>

#### Returns

The Shape of the Model or undefined.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Default value	Description
this	Type<T>	undefined	The explicit static polymorphic this parameter.
model	T	undefined	The Model which is to be <b>serialized</b> .
shallow	boolean	false	Whether to <b>serialize</b> the Model shallowly.

**Source**

packages/data/src/model/model.ts:683

**data.Model.****treemap**

(Static Method)

Static **treemap** method. Calling this method on a class extending the abstract Model base class, while supplying a model which to **treemap** and optionally enabling shallow **treemapping**, will return a Graph describing the fields which are declared and defined on the supplied model, or undefined, if the supplied model does not contain any fields or values. By **treemapping** shallowly, only properties defined on the supplied model are included (which means, all one-to-one and one-to-many associations are ignored). Through this method, the Graph for one specific Model instance from the respective Model repository can be retrieved.

**Example**

**treemap** a Model:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
const graph = ExampleModel.treemap(model);
console.log(graph); // ['field']
```

**Signature**

treemap<T>(this, model, shallow?): undefined | Graph<T>

**Returns**

The Graph of the Model or undefined.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Default value	Description
this	Type<T>	undefined	The explicit static polymorphic this parameter.
model	T	undefined	The Model which is to be <b>treemapped</b> .
shallow	boolean	false	Whether to <b>treemap</b> the Model shallowly.

**Source**

packages/data/src/model/model.ts:752

**data.Model.****unravel***(Static Method)*

Static **unravel** method. Calling this method on a class extending the abstract Model base class, while supplying a graph describing the fields which to **unravel**, will return the Graph as raw string. Through this method, the Graph for one specific Model instance from the respective Model repository can be **unraveled** into a raw string. This **unraveled** Graph can then be consumed by, e.g., the commit method.

**Example**

**unravel** a Graph:

```
import { ExampleModel } from './example-model';

const unraveled = ExampleModel.unravel([
  'uuid',
  'modified',
  'field'
]);

console.log(unraveled); // '{id modified field}'
```

**Signature**

unravel<T>(this, graph): string

**Returns**

The **unraveled** Graph as raw string.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
graph	Graph<T>	A Graph which is to be <b>unraveled</b> .

**Source**

packages/data/src/model/model.ts:817

**data.Model.****value***(Static Method)*

Static **value** method. Calling this method on a class extending the abstract Model base class, while supplying a model and a field which to **value**, will return the preprocessed value (e.g., primitive representation of JavaScript Dates) of the supplied field of the supplied model. Through this method, the preprocessed field value of one specific Model instance from the respective Model repository can be retrieved.

**Example**

**value** a field:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ created: new Date(0) });
const value = ExampleModel.value(model, 'created');
console.log(value); // '1970-01-01T00:00:00.000+00:00'
```

**Signature**

value<T>(this, model, field): unknown

**Returns**

The **valuated** field value.



**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
model field	T Field<T>	The Model which is to be <b>valuated</b> . A Field to be <b>valuated</b> .

**Source**

packages/data/src/model/model.ts:887

**data.Model.****[hasMany]**

(Optional Readonly Property)

hasMany symbol property used by the HasMany decorator.

**Source**

packages/data/src/model/model.ts:942

**data.Model.****[hasOne]**

(Optional Readonly Property)

hasOne symbol property used by the HasOne decorator.

**Source**

packages/data/src/model/model.ts:937

**data.Model.****[observable]**

(Method)

Well-known Symbol.observable method returning a Subscribable. The returned Subscribable emits all changes this Model instance experiences.

**Example**

Subscribe to a Model instance:

```
import { from } from 'rxjs';
import { ExampleModel } from './example-model';

const model = new ExampleModel();
from(model).subscribe(console.log);
```

**Signature**

[observable](): Subscribable<M>

**Returns**

A Subscribable emitting all Model changes.

**Source**

packages/data/src/model/model.ts:1045

**data.Model.****[property]***(Optional Readonly Property)*

property symbol property used by the Property decorator.

**Source**

packages/data/src/model/model.ts:947

**data.Model.****assign***(Method)*

Instance-scoped **assign** method. Calling this method, while supplying a list of parts, will **assign** all supplied parts to the Model instance. The **assignment** is implemented as deep merge **assignment**. Using this method, an existing Model instance can easily be mutated while still emitting the mutated changes.

**Example****assign** parts to a Model instance:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel();
model.assign({ field: 'example' }).subscribe(console.log);
```

**Signature**

assign&lt;T&gt;(this, ...parts): Observable&lt;T&gt;

**Returns**

An Observable of the mutated instance.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	T	The explicit polymorphic this parameter.
...parts	Shape<T>[]	An array of parts to <b>assign</b> to this Model.

**Source**

packages/data/src/model/model.ts:1070

**data.Model.****clear***(Method)*

Instance-scoped **clear** method. Calling this method on an instance of a class extending the abstract Model base class, while optionally supplying a list of keys which are to be **cleared**, will set the value of the properties described by either the supplied keys or, if no keys were supplied, all enumerable properties of the class extending the abstract Model base class to undefined, effectively **clearing** them.

**Example****clear** a Model instance selectively:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
model.clear(['field']).subscribe(console.log);
```

**Signature**

clear&lt;T&gt;(this, keys?): Observable&lt;T&gt;

**Returns**

An Observable of the mutated instance.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	T	The explicit polymorphic this parameter.
keys?	Field<T>[]	An optional array of keys to <b>clear</b> .

**Source**

packages/data/src/model/model.ts:1103

**data.Model.****commit**

(Method)

Instance-scoped **commit** method. Internally calls the commit method on the static **this**-context of an instance of a class extending the abstract Model base class and furthermore assigns the returned data to the Model instance the **commit** method was called upon. When supplying a mapping, the returned data will be mutated through the supplied mapping (otherwise this mapping defaults to identity).

**Example**

**commit** a query-type operation:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel();

model.commit(`query queryExample(variable: $variable) {
  result {
    field
  }
}`, {
  variable: 'value'
}).subscribe(console.log);
```

**Signature**

commit<T>(this, operation, variables?, mapping?): Observable<T>

**Returns**

An Observable of the mutated instance.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	T	The explicit polymorphic this parameter.
operation	Operation	The Operation to be <b>committed</b> .
variables?	Variables	Any Variables within the operation.
mapping	(next: unknown) => Shape<T>	An optional mutation to apply to the returned data.

**Source**

packages/data/src/model/model.ts:1160

**data.Model.****constructor***(Constructor)*

Public **constructor**. The **constructor** of all classes extending the abstract Model base class, unless explicitly overridden, behaves analogous to the instance-scoped assign method, as it takes all supplied parts and assigns them to the instantiated and returned Model. The **constructor** furthermore wires some internal functionality, e.g., creates a new changes BehaviorSubject which emits every mutation this Model instance experiences etc.

**Signature**

```
new Model<M>(...parts)
```

**Type parameters**

Name	Type
M	extends Model<any, M> = any

**Parameters**

Name	Type	Description
...parts	Shape<M>[]	An array of parts to assign.

**Source**

```
packages/data/src/model/model.ts:1022
```

**data.Model.****created***(Optional Property)*

Transient creation Date of this Model instance.

**Decorator**

Property

**Source**

```
packages/data/src/model/model.ts:963
```

**data.Model.****delete***(Method)*

Instance-scoped **delete** method. Internally calls the static deleteOne method while supplying the UUID of this instance of a class extending the abstract Model base class. Calling this method furthermore clears the Model instance and finalizes its deletion by completing the internal changes BehaviorSubject of the Model instance the **delete** method was called upon.

**Example**

**delete** a Model instance by UUID:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({
  id: '3068b30e-82cd-44c5-8912-db13724816fd'
});

model.delete().subscribe(console.log);
```

**Signature**

```
delete<T>(this): Observable<T>
```

**Returns**

An Observable of the mutated instance.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	T	The explicit polymorphic this parameter.

**Source**

packages/data/src/model/model.ts:1196

**data.Model.****find**

(Method)

Instance-scoped **find** method. Internally calls the findOne method on the static **this**-context of an instance of a class extending the abstract Model base class and then assigns the returned data to the Model instance the **find** method was called upon.

**Example**

**find** a Model instance by UUID:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({
  id: '3068b30e-82cd-44c5-8912-db13724816fd'
});

model.find([
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

**Signature**

find<T>(this, graph, shape?): Observable<T>

**Returns**

An Observable of the mutated instance.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	T	The explicit polymorphic this parameter.
graph	Graph<T>	A Graph of fields to be returned.
shape	Shape<T>	The Shape of the Model to find.

**Source**

packages/data/src/model/model.ts:1231

**data.Model.****modified**

(Optional Property)

Transient modification Date of this Model instance.

**Decorator**

Property

**Source**

packages/data/src/model/model.ts:971

**data.Model.****save***(Method)*

Instance-scoped **save** method. Internally calls the `saveOne` method on the static `this`-context of an instance of a class extending the abstract `Model` base class and then assigns the returned data to the `Model` instance the **save** method was called upon.

**Example**

**save** a `Model` instance:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });

model.save([
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

**Signature**

`save<T>(this, graph?): Observable<T>`

**Returns**

An `Observable` of the mutated instance.

**Type parameters**

Name	Type	Description
T	extends <code>Model&lt;any, T&gt; = M</code>	The extending <code>Model</code> InstanceType.

**Parameters**

Name	Type	Description
this	T	The explicit polymorphic <code>this</code> parameter.
graph	Graph<T>	A Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:1266

**data.Model.****serialize***(Method)*

Instance-scoped **serialize** method. Internally calls the `serialize` method on the static `this`-context of an instance of a class extending the abstract `Model` base class.

**Example**

**serialize** a `Model` instance:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
console.log(model.serialize()); // { field: 'example' }
```

**Signature**

```
serialize<T>(this, shallow?): undefined | Shape<T>
```

**Returns**

The Shape of this instance or undefined.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Default value	Description
this	T	undefined	The explicit polymorphic this parameter.
shallow	boolean	false	Whether to <b>serialize</b> shallowly.

**Source**

packages/data/src/model/model.ts:1294

**data.Model.****treemap**

(Method)

Instance-scoped **treemap** method. Internally calls the treemap method on the static this-context of an instance of a class extending the abstract Model base class.

**Example**

**treemap** a Model instance:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
console.log(model.treemap()); // ['field']
```

**Signature**

```
treemap<T>(this, shallow?): undefined | Graph<T>
```

**Returns**

A Graph of this instance or undefined.

**Type parameters**

Name	Type	Description
T	extends Model<any, T> = M	The extending Model InstanceType.

**Parameters**

Name	Type	Default value	Description
this	T	undefined	The explicit polymorphic this parameter.
shallow	boolean	false	Whether to <b>treemap</b> shallowly.

**Source**

packages/data/src/model/model.ts:1320

**data.Model.****uuid**

(Optional Property)

UUID of this Model instance.

**Decorator**

Property

**Source**

packages/data/src/model/model.ts:955

**data.Model.****[toStringTag]***(Protected Readonly Abstract Property)*

Enforced well-known `Symbol.toStringTag` property containing the singular name of this `Model`. The value of this property represents the repository which all instances of this `Model` are considered to belong to. In detail, the different operations committed through this `Model` are derived from this singular name (and the corresponding pluralized form).

**Example**

Provide a valid symbol property:

```
import { Model } from '@sgrud/data';

export class ExampleModel extends Model<ExampleModel> {

  protected [Symbol.toStringTag]: string = 'ExampleModel';

}
```

**Source**

packages/data/src/model/model.ts:932

**data.Model.****changes***(Protected Readonly Property)*

BehaviorSubject emitting every time this `Model` instance experiences **changes**.

**Source**

packages/data/src/model/model.ts:977

**data.Model.****entity***(Protected Accessor)*

Accessor to the singular name of this `Model`.

**Signature**

```
get entity(): string
```

**Returns**

The singular name of this `Model`.

**Source**

packages/data/src/model/model.ts:989

**data.Model.****plural***(Protected Accessor)*

Accessor to the **pluralized** name of this `Model`.

**Signature**

```
get plural(): string
```



**Returns**

The **pluralized** name of this Model.

**Source**

packages/data/src/model/model.ts:998

**data.Model.****static**

*(Protected Readonly Property)*

Type-asserted alias for the **static** Model context.

**Source**

packages/data/src/model/model.ts:982

**data.Model.****type**

*(Protected Accessor)*

Accessor to the raw name of this Model.

**Signature**

get type(): string

**Returns**

The raw name of this Model.

**Source**

packages/data/src/model/model.ts:1007

**data.****Model**

*(Namespace)*

The **Model** namespace contains types and interfaces used and intended to be used in conjunction with classes extending the abstract Model base class. All the types and interfaces within this namespace are only applicable to classes extending the abstract Model base class, as their generic type argument is always constrained to this abstract base class.

**See**

Model

**Source**

packages/data/src/model/model.ts:18, packages/data/src/model/model.ts:126, packages/data/src/model/model.ts:323

**data.Model.****Field**

*(Type alias)*

Type alias for all **Fields**, i.e., own enumerable properties (excluding internally used ones), of classes extending the abstract Model base class.

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Source**

packages/data/src/model/model.ts:28

**data.Model.****Filter***(Type alias)***Filter** type alias referencing the Params type.**See**

Params

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Source**

packages/data/src/model/model.ts:38, packages/data/src/model/model.ts:126

**data.Model.****Filter***(Namespace)*

The **Filter** namespace contains types and interfaces to be used when searching through the repositories of classes extending the abstract Model base class. All the interfaces within this namespace are only applicable to classes extending the abstract Model base class, as their generic type argument is always constrained to this abstract base class.

**See**

Model

**Source**

packages/data/src/model/model.ts:38, packages/data/src/model/model.ts:126

**data.Model.Filter.****Conjunction***(Type alias)*Type alias for a string union type of all possible **Conjunctions**, namely: 'AND', 'AND\_NOT', 'OR' and 'OR\_NOT'.**Source**

packages/data/src/model/model.ts:132

**data.Model.Filter.****Expression***(Interface)*

Interface describing the shape of an **Expression** which may be employed through the Params as part of a findAll. **Expressions** can either be the plain shape of an entity or compositions of multiple conjunctions.

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Source**

packages/data/src/model/model.ts:160

**data.Model.Filter.Expression.****conjunction***(Optional Readonly Property)*

**conjunction** of multiple filter Expressions requested data Models are matched against. The **conjunction** sibling parameter entity has to be undefined when supplying this parameter. By supplying filter Expressions, conjunct by specific Conjunction operators, fine-grained filter

operations can be compiled.

#### Type declaration

Name	Type	Description
operands	Expression<T>[]	List of Expressions which are logically combined through an operator. These Expressions may be nested and can be used to construct complex composite filter operations.
operator?	Conjunction	Conjunction <b>operator</b> used to logically combine all supplied operands.

#### Source

packages/data/src/model/model.ts:170

#### data.Model.Filter.Expression.

### entity

(Optional Readonly Property)

Shape the requested data Models are matched against. Supplying this parameter requires the conjunction sibling parameter to be undefined. By specifying the **entity** shape to match data Models against, simple filter operations can be compiled.

#### Type declaration

Name	Type	Description
operator?	Operator	Operator to use for matching.
path	Path<T, []>	Property <b>path</b> from within the data Model which to match against. The value which will be matched against has to be supplied through the value property.
value	unknown	Property <b>value</b> to match data Models against. The property path to this value has to be supplied through the path property.

#### Source

packages/data/src/model/model.ts:193

#### data.Model.Filter.

### Operator

(Type alias)

Type alias for a string union type of all possible **Operators**, namely: 'EQUAL', 'NOT\_EQUAL', 'LIKE', 'GREATER\_THAN', 'GREATER\_OR\_EQUAL', 'LESS\_THAN' and 'LESS\_OR\_EQUAL'.

#### Source

packages/data/src/model/model.ts:143

#### data.Model.Filter.

### Params

(Interface)

Interface describing the **Params** for the findAll method. This is the most relevant interface within this namespace (and is therefore also referenced by the Filter type alias), as it describes the input **Params** of any selective data retrieval.

#### See

Model

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Source**

packages/data/src/model/model.ts:228

**data.Model.Filter.Params.****dir**

*(Optional Readonly Property)*

Desired sorting **direction** of the requested data Models. To specify which field the results should be sorted by, the sort property must be supplied.

**Source**

packages/data/src/model/model.ts:235

**data.Model.Filter.Params.****expression**

*(Optional Readonly Property)*

Expression to evaluate results against. This **expression** may be a simple matching or more complex, conjunct and nested **expressions**.

**Source**

packages/data/src/model/model.ts:242

**data.Model.Filter.Params.****page**

*(Optional Readonly Property)*

**page** number, i.e., offset within the list of all results for a data Model request. This property should be used together with the page size property.

**Source**

packages/data/src/model/model.ts:249

**data.Model.Filter.Params.****search**

*(Optional Readonly Property)*

Free-text **search** field. This field overrides all expressions, as such that if this field contains a value, all expressions are ignored and only this free-text **search** filter is applied.

**Source**

packages/data/src/model/model.ts:256

**data.Model.Filter.Params.****size**

*(Optional Readonly Property)*

Page **size**, i.e., number of results which should be included within the response to a data Model request. This property should be used together with the page offset property.

**Source**

packages/data/src/model/model.ts:263

**data.Model.Filter.Params.****sort***(Optional Readonly Property)*

Property path used to determine the value which to **sort** the requested data Models by. This property should be used together with the sorting direction property.

**Source**

packages/data/src/model/model.ts:270

**data.Model.Filter.****Results***(Interface)*

Interface describing the shape of Filtered **Results**. When invoking the findAll method, an Observable of this interface shape is returned.

**Type parameters**

Name	Type
T	extends Model

**Source**

packages/data/src/model/model.ts:279

**data.Model.Filter.Results.****result***(Property)*

An array of Models representing the Filtered **results**.

**Source**

packages/data/src/model/model.ts:284

**data.Model.Filter.Results.****total***(Property)*

The **total** number of Results, useful for the implementation of a pageable representation of Filtered Results.

**Source**

packages/data/src/model/model.ts:290

**data.Model.****Graph***(Type alias)*

Mapped type to compile strongly typed **Graphs** of classes extending the abstract Model base class, while providing intellisense.

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Source**

packages/data/src/model/model.ts:46

**data.Model.****Path***(Type alias)*

Mapped type to compile strongly typed property **Paths** of classes extending the abstract Model base class, while providing intellisense.

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.
N	extends string[] = []	A string array type used to determine recursive depth.

**Source**

packages/data/src/model/model.ts:63

**data.Model.****Shape***(Type alias)*

Mapped type to compile strongly typed **Shapes** of classes extending the abstract Model base class, while providing intellisense.

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Source**

packages/data/src/model/model.ts:80

**data.Model.****Type***(Interface)*

Interface describing the **Type**, i.e., static constructable context, of classes extending the abstract Model base class.

**Type parameters**

Name	Type	Description
T	extends Model	The extending Model InstanceType.

**Hierarchy**

- Required<typeof Model>
  - **Type**

**Source**

packages/data/src/model/model.ts:97

**data.Model.Type.****commit***(Method)*

Static **commit** method. Calling this method on a class extending the abstract Model base class, while supplying an operation and all its embedded variables, will dispatch the Operation to the respective Model repository through the highest priority Querier or, if no Querier is compatible, an error is thrown. This method is the entry point for all Model-related data transferral and is internally called by all other distinct methods of the Model.

**Throws**

An Observable ReferenceError on incompatibility.

**Example**

**commit** a query-type operation:

```
import { ExampleModel } from './example-model';

ExampleModel.commit(`query queryExample(variable: $variable) {
  result {
    field
  }
}`, {
  variable: 'value'
}).subscribe(console.log);
```

**Signature**

commit<T>(this, operation, variables?): Observable<unknown>

**Returns**

An Observable of the **committed** operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
operation	Operation	The Operation to be <b>committed</b> .
variables?	Variables	Any Variables within the operation.

**Source**

packages/data/src/model/model.ts:357

**data.Model.Type.****constructor**

(Constructor)

Overridden and concretized constructor signature.

**Signature**

new Type(...args)

**Parameters**

Name	Type	Description
...args	Shape<Model<any>>[]	The default class constructor rest parameter.

**Source**

packages/data/src/model/model.ts:97

**data.Model.Type.****deleteAll**

(Method)

Static **deleteAll** method. Calling this method on a class extending the Model, while supplying an array of uuids, will dispatch the deletion of all Model instances identified by these UUIDs to the respective Model repository by internally calling commit with suitable arguments. Through this method, bulk-deletions from the respective Model repository can be achieved.

**Example**

Drop all model instances by UUIDs:

```
import { ExampleModel } from './example-model';
```

```
ExampleModel.deleteAll([
  'b050d63f-cede-46dd-8634-a80d0563ead8',
  'a0164132-cd9b-4859-927e-ba68bc20c0ae',
  'b3fca31e-95cd-453a-93ae-969d3b120712'
]).subscribe(console.log);
```

#### Signature

`deleteAll<T>(this, uuids): Observable<unknown>`

#### Returns

An Observable of the deletion.

#### Type parameters

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

#### Parameters

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
uuids	string[]	An array of uuids of Models to be deleted.

#### Source

packages/data/src/model/model.ts:410

### data.Model.Type.

## deleteOne

(Method)

Static **deleteOne** method. Calling this method on a class extending the Model, while supplying an uuid, will dispatch the deletion of the Model instance identified by this UUID to the respective repository by internally calling the commit operation with suitable arguments. Through this method, the deletion of a single Model instance from the respective Model repository can be achieved.

#### Example

Drop one model instance by UUID:

```
import { ExampleModel } from './example-model';
```

```
ExampleModel.deleteOne(
  '18f3aa99-afa5-40f4-90c2-71a2ecc25651'
).subscribe(console.log);
```

#### Signature

`deleteOne<T>(this, uuid): Observable<unknown>`

#### Returns

An Observable of the deletion.

#### Type parameters

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

#### Parameters

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.



Name	Type	Description
uuid	string	The uuid of the Model instance to be deleted.

**Source**

packages/data/src/model/model.ts:444

**data.Model.Type.****findAll**

(Method)

Static **findAll** method. Calling this method on a class extending the abstract Model base class, while supplying a filter to match Model instances by and a graph containing the fields to be included in the result, will dispatch a lookup operation to the respective repository by internally calling the commit operation with suitable arguments. Through this method, the bulk-lookup of Model instances from the respective Model repository can be achieved.

**Example**

Lookup all UUIDs for model instances modified between two dates:

```
import { ExampleModel } from './example-model';
```

```
ExampleModel.findAll({
  expression: {
    conjunction: {
      operands: [
        {
          entity: {
            operator: 'GREATER_OR_EQUAL',
            path: 'modified',
            value: new Date('2021-01-01')
          }
        },
        {
          entity: {
            operator: 'LESS_OR_EQUAL',
            path: 'modified',
            value: new Date('2021-12-12')
          }
        }
      ],
      operator: 'AND'
    }
  }, [
    'uuid',
    'field'
  ]).subscribe(console.log);
```

**Signature**

```
findAll<T>(this, filter, graph): Observable<Results<T>>
```

**Returns**

An Observable of the find operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
filter	Filter<T>	A Filter to find Model instances by.
graph	Graph<T>	A Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:503

**data.Model.Type.****findOne**

(Method)

Static **findOne** method. Calling this method on a class extending the abstract Model base class, while supplying the shape to match the Model instance by and a graph describing the fields to be included in the result, will dispatch the lookup operation to the respective repository by internally calling the commit operation with suitable arguments. Through this method, the retrieval of one specific Model instance from the respective Model repository can be achieved.

**Example**

Lookup one model instance by UUID:

```
import { ExampleModel } from './example-model';

ExampleModel.findOne({
  id: '2cfe7609-c4d9-4e4f-9a8b-ad72737db48a'
}, [
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

**Signature**

findOne<T>(this, shape, graph): Observable<T>

**Returns**

An Observable of the find operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
shape	Shape<T>	The Shape of instance to find.
graph	Graph<T>	A Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:552

**data.Model.Type.****prototype**

(Readonly Property)

Overridden prototype signature.

**Source**

packages/data/src/model/model.ts:102

**data.Model.Type.****saveAll**

(Method)

Static **saveAll** method. Calling this method on a class extending the abstract Model base class, while supplying a list of models which to save and a graph describing the fields to be returned in the result, will dispatch the save operation to the respective Model repository by internally

calling the commit operation with suitable arguments. Through this method, bulk-persistence of Model instances from the respective Model repository can be achieved.

#### Example

Persist multiple Models:

```
import { ExampleModel } from './example-model';

ExampleModel.saveAll([
  new ExampleModel({ field: 'example_1' }),
  new ExampleModel({ field: 'example_2' }),
  new ExampleModel({ field: 'example_3' })
], [
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

#### Signature

saveAll<T>(this, models, graph): Observable<T[]>

#### Returns

An Observable of the save operation.

#### Type parameters

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

#### Parameters

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
models	T[]	An array of Models to be saved.
graph	Graph<T>	The Graph of fields to be returned.

#### Source

packages/data/src/model/model.ts:598

#### data.Model.Type.

### saveOne

(Method)

Static **saveOne** method. Calling this method on a class extending the abstract Model base class, while supplying a model which to save and a graph describing the fields to be returned in the result, will dispatch the save operation to the respective Model repository by internally calling the commit operation with suitable arguments. Through this method, persistence of one specific Model instance from the respective Model repository can be achieved.

#### Example

Persist a model:

```
import { ExampleModel } from './example-model';

ExampleModel.saveOne(new ExampleModel({ field: 'example' }), [
  'uuid',
  'modified',
  'field'
]).subscribe(console.log);
```

#### Signature

saveOne<T>(this, model, graph): Observable<T>

#### Returns

An Observable of the save operation.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
model	T	The Model which is to be saved.
graph	Graph<T>	A Graph of fields to be returned.

**Source**

packages/data/src/model/model.ts:640

**data.Model.Type.****serialize**

(Method)

Static **serialize** method. Calling this method on a class extending the Model, while supplying a model which to **serialize** and optionally enabling shallow serialization, will return the **serialized** Shape of the Model, i.e., a plain JSON representation of all Model fields, or undefined, if the supplied model does not contain any fields or values. By serializing shallowly, only such properties defined on the supplied model are included (which means, all one-to-one and one-to-many associations are ignored). Through this method, the serialization of one specific Model instance from the respective Model repository can be achieved.

**Example**

**serialize** a model:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
const shape = ExampleModel.serialize(model);
console.log(shape); // { field: 'example' }
```

**Signature**

serialize<T>(this, model, shallow?): undefined | Shape<T>

**Returns**

The Shape of the Model or undefined.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Default value	Description
this	Type<T>	undefined	The explicit static polymorphic this parameter.
model	T	undefined	The Model which is to be <b>serialized</b> .
shallow	boolean	false	Whether to <b>serialize</b> the Model shallowly.

**Source**

packages/data/src/model/model.ts:683

**data.Model.Type.****treemap***(Method)*

Static **treemap** method. Calling this method on a class extending the abstract Model base class, while supplying a model which to **treemap** and optionally enabling shallow **treemapping**, will return a Graph describing the fields which are declared and defined on the supplied model, or undefined, if the supplied model does not contain any fields or values. By **treemapping** shallowly, only properties defined on the supplied model are included (which means, all one-to-one and one-to-many associations are ignored). Through this method, the Graph for one specific Model instance from the respective Model repository can be retrieved.

**Example**

**treemap** a Model:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ field: 'example' });
const graph = ExampleModel.treemap(model);
console.log(graph); // ['field']
```

**Signature**

treemap<T>(this, model, shallow?): undefined | Graph<T>

**Returns**

The Graph of the Model or undefined.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Default value	Description
this	Type<T>	undefined	The explicit static polymorphic this parameter.
model	T	undefined	The Model which is to be <b>treemapped</b> .
shallow	boolean	false	Whether to <b>treemap</b> the Model shallowly.

**Source**

packages/data/src/model/model.ts:752

**data.Model.Type.****unravel***(Method)*

Static **unravel** method. Calling this method on a class extending the abstract Model base class, while supplying a graph describing the fields which to **unravel**, will return the Graph as raw string. Through this method, the Graph for one specific Model instance from the respective Model repository can be **unraveled** into a raw string. This **unraveled** Graph can then be consumed by, e.g., the commit method.

**Example**

**unravel** a Graph:

```
import { ExampleModel } from './example-model';

const unraveled = ExampleModel.unravel([
  'uuid',
  'modified',
  'field'
]);

console.log(unraveled); // '{id modified field}'
```

**Signature**

unravel<T>(this, graph): string

**Returns**

The **unraveled** Graph as raw string.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
graph	Graph<T>	A Graph which is to be <b>unraveled</b> .

**Source**

packages/data/src/model/model.ts:817

**data.Model.Type.****value**

(Method)

Static **value** method. Calling this method on a class extending the abstract Model base class, while supplying a **model** and a **field** which to **value**, will return the preprocessed value (e.g., primitive representation of JavaScript Dates) of the supplied **field** of the supplied **model**. Through this method, the preprocessed **field** value of one specific Model instance from the respective Model repository can be retrieved.

**Example**

**value** a field:

```
import { ExampleModel } from './example-model';

const model = new ExampleModel({ created: new Date(0) });
const value = ExampleModel.value(model, 'created');
console.log(value); // '1970-01-01T00:00:00.000+00:00'
```

**Signature**

value<T>(this, model, field): unknown

**Returns**

The **valuated** field value.

**Type parameters**

Name	Type	Description
T	extends Model<any, T>	The extending Model InstanceType.

**Parameters**

Name	Type	Description
this	Type<T>	The explicit static polymorphic this parameter.
model	T	The Model which is to be <b>valuated</b> .
field	Field<T>	A Field to be <b>valuated</b> .

**Source**

packages/data/src/model/model.ts:887

**data.****Property**

(Type alias)

Type alias for a union type of all primitive constructors which may be used as **typeFactory** argument for the Property decorator.

**See**

Property

**Source**

packages/data/src/relation/property.ts:61, packages/data/src/relation/property.ts:10

**data.**

## Property

*(Function)*

Model field decorator factory. Using this decorator, Models can be enriched with primitive fields. The compatible primitives are the subset of primitives JavaScript shares with JSON, i.e., Boolean, Date (serialized), Number and String. Objects cannot be used as a `typeFactory` argument value, as Model fields containing objects should be declared by the `HasOne` and `HasMany` decorators. By employing this decorator, the decorated field will (depending on the `transient` argument value) be taken into account when serializing or treemapping the Model containing the decorated field.

**Example**

Model with a primitive field:

```
import { Model, Property } from '@sgrud/data';

export class ExampleModel extends Model<ExampleModel> {

  @Property(() => String)
  public field?: string;

  protected [Symbol.toStringTag]: string = 'ExampleModel';
}
```

**See**

Model, HasOne, HasMany

**Signature**

```
Property<T>(typeFactory, transient?): <M>(model: M, field: Field<M>) => void
```

**Returns**

A Model field decorator.

**Type parameters**

Name	Type	Description
T	extends Property	The field value constructor type.

**Parameters**

Name	Type	Default value	Description
typeFactory	() => T	undefined	A forward reference to the field value constructor.
transient	boolean	false	Whether the decorated field is transient.

**Source**

packages/data/src/relation/property.ts:61, packages/data/src/relation/property.ts:10

**data.**

## Querier

*(Abstract Class)*

Abstract **Querier** base class to implement Model **Queriers**. By extending this abstract base class and providing the extending class to the Linker, e.g., by Targeting it, the `priority` method of the resulting class will be called whenever the Model requests or persists data and, if this class claims the highest priority, its `commit` method will be called.

**Decorator**

Provide

**Example**Simple **Querier** stub:

```
import { Provider, Target } from '@sgrud/core';
import { type Querier } from '@sgrud/data';
import { type Observable } from 'rxjs';

@Target()
export class ExampleQuerier
  extends Provider<typeof Querier>('sgrud.data.Querier') {

  public override readonly types: Set<Querier.Type> = new Set<Querier.Type>([
    'query'
  ]);

  public override commit(
    operation: Querier.Operation,
    variables: Querier.Variables
  ): Observable<unknown> {
    throw new Error('Stub!');
  }

  public override priority(): number {
    return 0;
  }
}
```

**See**

Model

**Hierarchy**

- **Querier**
  - BusQuerier
  - HttpQuerier

**Source**

packages/data/src/querier/querier.ts:12, packages/data/src/querier/querier.ts:89

**data.Querier.****[provide]***(Static Readonly Property)*

Magic string by which this class is provided.

**See**

provide

**Source**

packages/data/src/querier/querier.ts:96

**data.Querier.****commit***(Abstract Method)*

The overridden **commit** method of Targeted Queriers is called by the Model to execute Operations. The invocation arguments are the operation, unraveled into a string, and all variables embedded within this operation. The extending class has to serialize the Variables and handle the operation. It's the callers responsibility to unravel the Operation prior to invoking this method, and to deserialize and (error) handle whatever response is received.

**Signature**

commit(operation, variables?): Observable&lt;unknown&gt;



**Returns**

An Observable of the **committed** Operation.

**Parameters**

Name	Type	Description
operation	Operation	The Operation to be <b>committed</b> .
variables?	Variables	Any Variables within the Operation.

**Source**

packages/data/src/querier/querier.ts:118

**data.Querier.****priority**

*(Abstract Method)*

When the Model executes Operations, all Targeted and compatible Queriers, i.e., implementations of the this class capable of handling the specific Type of the Operation to commit, will be asked to prioritize themselves regarding the respective Model. The querier claiming the highest **priority** will be chosen and its commit method called.

**Signature**

priority(model): number

**Returns**

The numeric **priority** of this Querier implementation.

**Parameters**

Name	Type	Description
model	Type<Model<any>>	The Model to be committed.

**Source**

packages/data/src/querier/querier.ts:134

**data.Querier.****types**

*(Readonly Abstract Property)*

A set containing all **types** of queries this Querier can handle. May contain any of the 'mutation', 'query' and 'subscription' Types.

**Source**

packages/data/src/querier/querier.ts:103

**data.****Querier**

*(Namespace)*

**Querier** namespace containing types and interfaces used and intended to be used in conjunction with the abstract Querier base class and in context of the Model data handling.

**See**

Querier

**Source**

packages/data/src/querier/querier.ts:12, packages/data/src/querier/querier.ts:89

**data.Querier.**

## Operation

*(Type alias)*

String literal helper type. Enforces any assigned string to conform to the standard form of an **Operation**: A string starting with the Type, followed by one whitespace and the operation content.

**Source**

packages/data/src/querier/querier.ts:28

**data.Querier.**

## Type

*(Type alias)*

Type alias for a string union type of all known Operation **Types**: 'mutation', 'query' and 'subscription'.

**Source**

packages/data/src/querier/querier.ts:18

**data.Querier.**

## Variables

*(Interface)*

Interface describing the shape of **Variables** which may be embedded within Operations. **Variables** are a simple key-value map, which can be deeply nested.

**Source**

packages/data/src/querier/querier.ts:35

**data.**

## enumerate

*(Function)*

**enumerate** helper function. Enumerations are special objects and all used TypeScript enums have to be looped through this helper function before they can be utilized in conjunction with the Model.

**Example**

**enumerate** a TypeScript enumeration:

```
import { enumerate } from '@sgrud/data';

enum Enumeration {
  One = 'ONE',
  Two = 'TWO'
}

export type ExampleEnum = Enumeration;
export const ExampleEnum = enumerate(Enumeration);
```

**See**

Model

**Signature**

enumerate<T>(enumerator): T

**Returns**

The processed enumeration to be used by the Model.

**Type parameters**

Name	Type	Description
T	extends object	The type of TypeScript enum.

**Parameters**

Name	Type	Description
enumerator	T	The TypeScript enum to <b>enumerate</b> .

**Source**

packages/data/src/model/enum.ts:49

**data.****hasMany**

*(Const Variable)*

Unique symbol used as property key by the HasMany decorator to register decorated Model fields for further computation, e.g., serialization, treemapping etc.

**See**

HasMany

**Source**

packages/data/src/relation/has-many.ts:11

**data.****hasOne**

*(Const Variable)*

Unique symbol used as property key by the HasOne decorator to register decorated Model fields for further computation, e.g., serialization, treemapping etc.

**See**

HasOne

**Source**

packages/data/src/relation/has-one.ts:11

**data.****property**

*(Const Variable)*

Unique symbol used as property key by the Property decorator to register decorated Model fields for further computation, e.g., serialization, treemapping etc.

**See**

Property

**Source**

packages/data/src/relation/property.ts:24

**@sgrud/shell Module**

**@sgrud/shell** - The SGRUD Web UI Shell.

The functions and classes found within the **@sgrud/shell** module are intended to ease the implementation of Component-based frontends by providing JSX runtime bindings via the **@sgrud/shell/jsx-runtime** module for the incremental-dom library and the Router to enable routing through Components based upon the SGRUD client libraries, but not limited to those. Furthermore, complex routing strategies and actions may be implemented through the interceptor-like Queue pattern.

**Source**

packages/shell/index.ts:1

shell.

## Attribute

(Function)

Component prototype property decorator factory. Applying the **Attribute** decorator to a property of a Component binds the decorated property to the corresponding **Attribute** of the respective Component. This implies that the **Attribute** name is appended to the observedAttributes array of the Component and the decorated property is replaced with a getter and setter deferring those operations to the **Attribute**. If no name supplied, the name of the decorated property will be used instead. Further, if both, a parameter initializer and an initial **Attribute** value are supplied, the **Attribute** value takes precedence.

### Example

Bind a property to an **Attribute**:

```
import { Attribute, Component } from '@sgrud/shell';

declare global {
  interface HTMLElementTagNameMap {
    'example-component': ExampleComponent;
  }
}

@Component('example-component')
export class ExampleComponent extends HTMLElement implements Component {

  @Attribute()
  public field?: string;

  public get template(): JSX.Element {
    return <span>Attribute value: {this.field}</span>;
  }
}
```

### See

Component

### Signature

Attribute(name?): (prototype: Component, propertyKey: PropertyKey) => void

### Returns

A Component prototype property decorator.

### Parameters

Name	Type	Description
name?	string	The Component <b>Attribute</b> name.

### Source

packages/shell/src/component/attribute.ts:45

shell.

## Catch

(Type alias)

The **Catch** type alias is used and intended to be used in conjunction with the CatchQueue and represents a function that is called with the thrown error. The return value of this callback will be used to examine whether the Component containing the decorated property is responsible to handle the thrown error.

### See

CatchQueue

### Source

packages/shell/src/queue/catch.ts:61, packages/shell/src/queue/catch.ts:17

shell.

## Catch

(Function)

Component prototype property decorator factory. Applying the **Catch** decorator to a property, while optionally supplying a **trap** will navigate to the Component containing the decorated property when an error, **traped** by this **Catch** decorator, occurs during navigation.

### Example

**Catch** all URIErrors:

```
import { Component, Catch } from '@sgrud/shell';

declare global {
  interface HTMLElementTagNameMap {
    'example-component': ExampleComponent;
  }
}

@Component('example-component')
export class ExampleComponent extends HTMLElement implements Component {

  @Catch((error) => error instanceof URIError)
  public readonly error?: URIError;

  public get template(): JSX.Element {
    return <span>Error message: {this.error?.message}</span>;
  }
}
```

### See

CatchQueue

### Signature

Catch(trap?): (prototype: Component, propertyKey: PropertyKey) => void

### Returns

A Component prototype property decorator.

### Parameters

Name	Type	Description
trap?	Catch	The Catch callback deciding whether to trap an error.

### Source

packages/shell/src/queue/catch.ts:61, packages/shell/src/queue/catch.ts:17

shell.

## CatchQueue

(Class)

This built-in **CatchQueue** extension of the Queue base class is used by the Catch decorator to intercept Router navigation events and handles all errors thrown during the asynchronous evaluation of navigate invocations. When the Catch decorator is applied at least once this **CatchQueue** will be automatically provided as Queue to the Linker-

### Decorator

Singleton

### See

Queue

### Hierarchy

- Queue<this>
  - **CatchQueue**

**Source**

packages/shell/src/queue/catch.ts:117

**shell.CatchQueue.****[provide]**

*(Static Readonly Property)*

Magic string by which this class is provided.

**See**

provide

**Source**

packages/shell/src/queue/queue.ts:49

**shell.CatchQueue.****constructor**

*(Constructor)*

Public Singleton **constructor**. Called by the Catch decorator to link this Queue into the Router and to access the trapped and traps properties.

**Signature**

new CatchQueue()

**Source**

packages/shell/src/queue/catch.ts:145

**shell.CatchQueue.****handle**

*(Method)*

Overridden **handle** method of the Queue base class. Iterates all Segments of the next State and collects all traps for any encountered Components in those iterated Segments.

**Signature**

handle(\_prev, next, queue): Observable<State<string>>

**Returns**

An Observable of the **handled** State.

**Parameters**

Name	Type	Description
_prev	State<string>	The _previously active State (ignored).
next	State<string>	The next State navigated to.
queue	Queue	The next Queue to <b>handle</b> the navigation.

**Source**

packages/shell/src/queue/catch.ts:163

**shell.CatchQueue.****trapped**

*(Readonly Property)*

Mapping of all decorated Components to a Map of property keys and **trapped** errors.

**Source**

packages/shell/src/queue/catch.ts:124

**shell.CatchQueue.****traps***(Readonly Property)*Mapping of all decorated Components to a Map of property keys and their **traps**.**Source**

packages/shell/src/queue/catch.ts:130

**shell.CatchQueue.****handleErrors***(Private Method)***handleErrors** helper method returning an Observable from the global `window.onerror` and `window.unhandledrejection` event emitters. The returned Observable will either NEVER complete or invoke `throwError` with any globally emitted `ErrorEvent` or the reason for a `PromiseRejectionEvent` while subscribed to.**Throws**

An Observable of any globally emitted error or rejection.

**Signature**`handleErrors(): Observable<never>`**Returns**

An Observable that NEVER completes.

**Source**

packages/shell/src/queue/catch.ts:260

**shell.CatchQueue.****router***(Private Readonly Property)*Factored-in **router** property linking the Router.**Decorator**

Factor

**Source**

packages/shell/src/queue/catch.ts:138

**shell.****Component***(Function)*Class decorator factory. Registers the decorated class as **Component** through the `customElements` registry. Registered **Components** can be used in conjunction with any of the `Attribute`, `Fluctuate` and `Reference` prototype property decorators which will trigger their respective callbacks or `renderComponent` whenever one of the `observedAttributes`, `observedFluctuations` or `observedReferences` changes. While any **Component** registered by this decorator is enriched with basic rendering functionality, any implemented method will cancel out its super logic.**Example**Register a **Component**:

```
import { Component } from '@sgrud/shell';

declare global {
  interface HTMLElementTagNameMap {
    'example-component': ExampleComponent;
  }
}

@Component('example-component')
export class ExampleComponent extends HTMLElement implements Component {
```

```

public readonly styles: string[] = [
  span {
    font-style: italic;
  }
];

public get template(): JSX.Element {
  return <span>Example component</span>;
}
}

```

**See**

Attribute, Reference

**Signature**

Component<S, K>(selector, inherits?): <T>(constructor: T) => T

**Returns**

A class constructor decorator.

**Type parameters**

Name	Type	Description
S	extends CustomElementTagName	The custom <b>Component</b> tag name selector type.
K	extends HTMLElementTagName	-

**Parameters**

Name	Type	Description
selector	S	The custom <b>Component</b> tag name selector.
inherits?	K	The HTMLElement this <b>Component</b> inherits from.

**Source**

packages/shell/src/component/component.ts:154, packages/shell/src/component/component.ts:16

**shell.****Component**

(Interface)

An interface describing the shape of a **Component**. Mostly adheres to the Web Components specification while providing rendering and change detection capabilities.

**Hierarchy**

- HTMLElement
  - **Component**

**Source**

packages/shell/src/component/component.ts:154, packages/shell/src/component/component.ts:16

**shell.Component.****adoptedCallback**

(Optional Method)

Called when the Component is moved between Documents.

**Signature**

adoptedCallback(): void



**Source**

packages/shell/src/component/component.ts:52

**shell.Component.****attributeChangedCallback**

*(Optional Method)*

Called when one of the Component's observed Attributes is added, removed or changed. Which Component attributes are observed depends on the contents of the observedAttributes array.

**Signature**

attributeChangedCallback(name, prev?, next?): void

**Parameters**

Name	Type	Description
name	string	The name of the changed attribute.
prev?	string	The previous value of the changed attribute.
next?	string	The next value of the changed attribute.

**Source**

packages/shell/src/component/component.ts:63

**shell.Component.****connectedCallback**

*(Optional Method)*

Called when the Component is appended to the Document.

**Signature**

connectedCallback(): void

**Source**

packages/shell/src/component/component.ts:68

**shell.Component.****disconnectedCallback**

*(Optional Method)*

Called when the Component is removed from the Document.

**Signature**

disconnectedCallback(): void

**Source**

packages/shell/src/component/component.ts:73

**shell.Component.****fluctuationChangedCallback**

*(Optional Method)*

This callback is invoked whenever a Component Fluctuates, i.e., if the any of its decorated propertyKeys is assigned the next value emitted by one of the observedFluctuations.

**Signature**

fluctuationChangedCallback(propertyKey, prev, next): void

**Parameters**

Name	Type	Description
propertyKey	PropertyKey	The propertyKey that Fluctuated.
prev	unknown	-
next	unknown	The previous value of the Fluctuated propertyKey.

**Source**

packages/shell/src/component/component.ts:84

**shell.Component.****observedAttributes**

*(Optional Readonly Property)*

Array of Attribute names, which should be observed for changes, which will trigger the attributeChangedCallback.

**Source**

packages/shell/src/component/component.ts:22

**shell.Component.****observedFluctuations**

*(Optional Readonly Property)*

A Record of Subscriptions opened by the Fluctuate decorator which trigger the fluctuationChangedCallback upon each emission, while subscribed to.

**Source**

packages/shell/src/component/component.ts:29

**shell.Component.****observedReferences**

*(Optional Readonly Property)*

A Record of References and observed events, which, when emitted by the reference, trigger the referenceChangedCallback.

**Source**

packages/shell/src/component/component.ts:35

**shell.Component.****referenceChangedCallback**

*(Optional Method)*

Called when one of the Component's Referenced and observed nodes emits an event. Which Referenced nodes are observed for which events depends on the contents of the observedReferences mapping.

**Signature**

referenceChangedCallback(key, node, event): void

**Parameters**

Name	Type	Description
key	Key	The key used to Reference the node.
node	Node	The Referenced node.
event	Event	The event emitted by the node.

**Source**

packages/shell/src/component/component.ts:99

**shell.Component.****renderComponent***(Optional Method)*

Called when the Component has changed and should render.

**Signature**`renderComponent(): void`**Source**

packages/shell/src/component/component.ts:104

**shell.Component.****styles***(Optional Readonly Property)*Array of CSS **styles** in string form, which should be included within the ShadowRoot of the Component.**Source**

packages/shell/src/component/component.ts:41

**shell.Component.****template***(Optional Readonly Property)*JSX representation of the Component **template**. If no template is supplied, an HTMLSlotElement will be rendered instead.**Source**

packages/shell/src/component/component.ts:47

**shell.****CustomElementTagName***(Type alias)*String literal helper type. Enforces any assigned string to be a keyof HTMLElementTagNameMap, while excluding built-in tag names, i.e., extracting ``${string}-${string}`` keys of the HTMLElementTagNameMap.**Example**A valid **CustomElementTagName**:

```
const tagName: CustomElementTagName = 'example-component';
```

**Source**

packages/shell/src/component/runtime.ts:18

**shell.****Fluctuate***(Function)*Component prototype property decorator factory. Applying this **Fluctuate** decorator to a property of a custom Component while supplying a **streamFactory** that returns an ObservableInput upon invocation will subscribe the fluctuationChangedCallback method to each emission from this ObservableInput and replace the decorated property with a getter returning its last emitted value. Further, the resulting subscription, referenced by the decorated property, is assigned to the observedFluctuations property and may be terminated by unsubscribing manually. Finally, the Component will seize to **Fluctuate** automatically when it's disconnected from the Document.**Example**A Component that **Fluctuates**:

```
import { Component, Fluctuate } from '@sgrud/shell';
import { fromEvent } from 'rxjs';
```

```
declare global {
  interface HTMLElementTagNameMap {
```

```

    'example-component': ExampleComponent;
  }
}

@Component('example-component')
export class ExampleComponent extends HTMLElement implements Component {

  @Fluctuate(() => fromEvent(document, 'click'))
  private readonly pointer?: MouseEvent;

  public get template(): JSX.Element {
    return <span>Clicked at ({this.pointer?.x}, {this.pointer?.y})</span>;
  }
}

```

**See**

Component

**Signature**

Fluctuate(streamFactory): (prototype: Component, propertyKey: PropertyKey) => void

**Returns**

A Component prototype property decorator.

**Parameters**

Name	Type	Description
streamFactory	() => ObservableInput<unknown>	A forward reference to an ObservableInput.

**Source**

packages/shell/src/component/fluctuate.ts:47

**shell.****HTMLElementTagName**

(Type alias)

String literal helper type. Enforces any assigned string to be a keyof HTMLElementTagNameMap, while excluding custom element tag names, i.e., \${string}-\${string} keys of the HTMLElementTagNameMap.

**Example**

A valid **HTMLElementTagName**:

```
const tagName: HTMLElementTagName = 'div';
```

**Source**

packages/shell/src/component/runtime.ts:32

**shell.****JSX**

(Namespace)

The intrinsic JSX namespace used by TypeScript to determine the Element type and all valid IntrinsicElements.

**Source**

packages/shell/src/component/runtime.ts:40

**shell.JSX.****Element**

(Type alias)

Intrinsic JSX **Element** type helper representing an array of bound elementOpen and elementClose calls.

**Source**

packages/shell/src/component/runtime.ts:46

**shell.JSX.****IntrinsicElements**

(Type alias)

List of known JSX **IntrinsicElements**, comprised of the global HTMLTagNameMap.

**Source**

packages/shell/src/component/runtime.ts:52

**shell.JSX.****Key**

(Type alias)

**Key** references type helper. Enforces any assigned values to be of a compatible **Key** type.

**Source**

packages/shell/src/component/runtime.ts:84

**shell.****Queue**

(Abstract Class)

Abstract base class to implement Router **Queues**. By applying the Target decorator or otherwise providing an implementation of this abstract **Queue** base class to the Linker, the implemented handle method is called whenever a new State is triggered by navigating. This interceptor-like pattern makes complex routing strategies like asynchronous module-retrieval and the similar tasks easy to be implemented.

**Decorator**

Provide

**Example**

Simple **Queue** stub:

```
import { Provider, Target } from '@sgrud/core';
import { type Router, type Queue } from '@sgrud/shell';
import { type Observable } from 'rxjs';

@Target()
export class ExampleQueue
  extends Provider<typeof Queue>('sgrud.shell.Queue') {

  public override handle(
    prev: Router.State,
    next: Router.State,
    queue: Router.Queue
  ): Observable<Router.State> {
    throw new Error('Stub!');
  }
}
```

**See**

Route, Router

**Hierarchy**

- **Queue**
  - CatchQueue
  - ResolveQueue

**Source**

packages/shell/src/queue/queue.ts:42

**shell.Queue.****[provide]***(Static Readonly Property)*

Magic string by which this class is provided.

**See**

provide

**Source**

packages/shell/src/queue/queue.ts:49

**shell.Queue.****handle***(Abstract Method)*

Abstract **handle** method, called whenever a new State should be navigated to. This method provides the possibility to intercept these upcoming States and, e.g., mutate or redirect them, i.e., **handle** the navigation.

**Signature**

handle(prev, next, queue): Observable&lt;State&lt;string&gt;&gt;

**Returns**An Observable of the **handled** State.**Parameters**

Name	Type	Description
prev	State<string>	The previously active State.
next	State<string>	The next State navigated to.
queue	Queue	The next Queue to <b>handle</b> the navigation.

**Source**

packages/shell/src/queue/queue.ts:62

**shell.****Reference***(Function)*

Component prototype property decorator factory. Applying this **Reference** decorator to a property of a registered Component while supplying the referenceing Key] and, optionally, an array of event names to observe, will replace the decorated property with a getter returning the referenced node, once rendered. If an array of event names is supplied, whenever one of those observed events is emitted by the referenced node, the referenceChangedCallback of the Component is called with the reference key, the referenced node and the emitted event.

**Example****Reference** a node:

```
import { Component, Reference } from '@sgrud/shell';

declare global {
  interface HTMLTagNameMap {
    'example-component': ExampleComponent;
  }
}

@Component('example-component')
export class ExampleComponent extends HTMLElement implements Component {

  @Reference('example-key')
  private readonly span?: HTMLSpanElement;

  public get template(): JSX.Element {
    return <span key="example-key"></span>;
  }
}
```

**See**

Component

**Signature**

Reference(reference, observe?): (prototype: Component, propertyKey: PropertyKey) => void

**Returns**

A Component prototype property decorator.

**Parameters**

Name	Type	Description
reference	Key	The referenceing Key.
observe?	keyof HTMLElementEventMap[]	An array of event names to observe.

**Source**

packages/shell/src/component/reference.ts:48

shell.

## Resolve

(Type alias)

The **Resolve** type alias is used and intended to be used in conjunction with the ResolveQueue Queue and the Resolve decorator. The **Resolve** type alias represents a function that will be called with the respective Segment and State.

**See**

Resolve

**Signature**

(segment, state): ObservableInput<unknown>

**Parameters**

Name	Type
segment	Segment<S>
state	State<S>

**Type parameters**

Name	Type	Description
S	extends string	The Route path string type.

**Source**

packages/shell/src/queue/resolve.ts:71, packages/shell/src/queue/resolve.ts:18

shell.

## Resolve

(Function)

Component prototype property decorator factory. Applying the **Resolve** decorator to a property of a Component, while supplying an ObservableInput to be resolved, will replace the decorated property with a getter returning the **Resolved** value the supplied ObservableInput resolves to. To do so the **Resolve** decorator relies on the built-in ResolveQueue.

**Example**

**Resolve** the Segment and State:

```
import { Component, Resolve } from '@sgrud/shell';
import { of } from 'rxjs';

declare global {
  interface HTMLElementTagNameMap {
```

```

    'example-component': ExampleComponent;
  }
}

@Component('example-component')
export class ExampleComponent extends HTMLElement implements Component {

  @Resolve((segment, state) => of([segment.route.path, state.search]))
  public readonly resolved!: [string, string];

  public get template(): JSX.Element {
    return <span>Resolved: {this.resolved.join('?')}</span>;
  }
}

```

**See**

ResolveQueue

**Signature**

Resolve&lt;S&gt;(resolve): (prototype: Component, propertyKey: PropertyKey) =&gt; void

**Returns**

A Component prototype property decorator.

**Type parameters**

Name	Type	Description
S	extends string	The Route path string type.

**Parameters**

Name	Type	Description
resolve	Resolve<S>	An ObservableInput to resolve.

**Source**

packages/shell/src/queue/resolve.ts:71, packages/shell/src/queue/resolve.ts:18

**shell.**

## ResolveQueue

*(Class)*

This built-in **ResolveQueue** extension of the Queue base class intercepts all navigational events of the Router to Resolve ObservableInputs before invoking subsequent Queues. Thereby this **ResolveQueue** allows asynchronous evaluations to be executed and their Resolved values to be provided to a Component, before it is rendered into a Document for the first time. When the Catch decorator is applied at least once this **ResolveQueue** will be automatically provided as Queue to the Linker.

**Decorator**

Singleton

**See**

Queue

**Hierarchy**

- Queue<this>
  - **ResolveQueue**

**Source**

packages/shell/src/queue/resolve.ts:129



**shell.ResolveQueue.****[provide]***(Static Readonly Property)*

Magic string by which this class is provided.

**See**

provide

**Source**

packages/shell/src/queue/queue.ts:49

**shell.ResolveQueue.****constructor***(Constructor)*

Public Singleton **constructor**. Called by the Resolve decorator to link this Queue into the Router and to access the required and resolved properties.

**Signature**

```
new ResolveQueue()
```

**Source**

packages/shell/src/queue/resolve.ts:149

**shell.ResolveQueue.****handle***(Method)*

Overridden **handle** method of the Queue base class. Iterates all Segments of the next State and collects all Resolvers for any encountered Components in those iterated Segments. The collected Resolvers are run before invoking the subsequent Queue.

**Signature**

```
handle(_prev, next, queue): Observable<State<string>>
```

**Returns**

An Observable of the **handled** State.

**Parameters**

Name	Type	Description
_prev	State<string>	The _previously active State (ignored).
next	State<string>	The next State navigated to.
queue	Queue	The next Queue to <b>handle</b> the navigation.

**Source**

packages/shell/src/queue/resolve.ts:168

**shell.ResolveQueue.****required***(Readonly Property)*

Mapping of all decorated Components to a Map of property keys and their **required** Resolvers.

**Source**

packages/shell/src/queue/resolve.ts:136

**shell.ResolveQueue.****resolved***(Readonly Property)*

Mapping of all decorated Components to an object consisting of property keys and their corresponding Resolved return values.

**Source**

packages/shell/src/queue/resolve.ts:142

shell.

**Route**

(Function)

Class decorator factory. Applying the **Route** decorator to a custom element will associate the supplied config with the decorated element constructor. Further, the configured children are iterated over and every child that is a custom element itself will be replaced by its respective route configuration or ignored, if no configuration was associated with the child. Finally, the processed config is added to the Router.

**Example**

Associate a Route config to a Component:

```
import { Component, Route } from '@sgrud/shell';
import { ChildComponent } from './child-component';

@Route({
  path: 'example',
  children: [
    ChildComponent
  ]
})
@Component('example-element')
export class ExampleComponent extends HTMLElement implements Component {}
```

**See**

Router

**Signature**

Route<S>(config): <T>(constructor: T) => void

**Returns**

A class constructor decorator.

**Type parameters**

Name	Type	Description
S	extends string	The Route path string type.

**Parameters**

Name	Type	Description
config	Assign<{ children?: (Route<string>   CustomElementConstructor & { [route]?: Route<string> }[])[]; slots?: Record<string, CustomElementConstructor   CustomElementTagName> }, Omit<Route<S>, "component">> & { parent?: Route<string>   CustomElementConstructor & { [route]?: Route<string> } }	The Route config for this element.

**Source**

packages/shell/src/router/route.ts:94, packages/shell/src/router/route.ts:33

shell.

**Route**

(Interface)

Interface describing the shape of a **Route**. A **Route** must consist of at least a path and may specify a component, as well as slots, which will be rendered into the RouterOutlet when the **Route** is navigated to. Furthermore a **Route** may also specify children.

**Example**

Define a **Route**:

```
import { type Route } from '@sgrud/shell';

const route: Route = {
  path: '',
  component: 'example-element',
  children: [
    {
      path: 'child',
      component: 'child-element'
    }
  ]
};
```

**See**

Router

**Type parameters**

Name	Type	Description
S	extends string = string	The <b>Route</b> path string type.

**Source**

packages/shell/src/router/route.ts:94, packages/shell/src/router/route.ts:33

**shell.Route.****children**

*(Optional Readonly Property)*

Optional array of **children** for this Route.

**Source**

packages/shell/src/router/route.ts:38

**shell.Route.****component**

*(Optional Readonly Property)*

Optional Route **component**.

**Source**

packages/shell/src/router/route.ts:43

**shell.Route.****path**

*(Readonly Property)*

Required Route **path**.

**Source**

packages/shell/src/router/route.ts:48

**shell.Route.****slots**

*(Optional Readonly Property)*

Optional mapping of elements to their **slots**.

**Source**

packages/shell/src/router/route.ts:53

**shell.**

## Router

*(Class)*

Targeted Singleton **Router** class extending the built-in Set. This Singleton class provides routing and rendering capabilities. Routing is primarily realized by maintaining the inherited Set of Routes and (recursively) matching paths against those Routes, when instructed so by the `navigate` method. When a matching Segment is found, the corresponding Components are rendered by the `handle` method (which is part of the implemented Queue contract).

**Decorator**

Target, Singleton

**Hierarchy**

- Set<Route>
- Router

**Implements**

Queue

**Source**

packages/shell/src/router/router.ts:13, packages/shell/src/router/router.ts:165

**shell.Router.**

## [observable]

*(Static Method)*

Static `Symbol.observable` method returning a `Subscribable`. The returned `Subscribable` mirrors the private loader and is used for initializations after a new global Route tree was added to the Router.

**Example**

Subscribe to the Router:

```
import { Router } from '@sgrud/shell';
import { from } from 'rxjs';

from(Router).subscribe(console.log);
```

**Signature**`[observable](): Subscribable<Router>`**Returns**A `Subscribable` emitting this Router.**Source**

packages/shell/src/router/router.ts:192

**shell.Router.**

## loader

*(Static Private Property)*

Private static `ReplaySubject` used as the Router **loader**. This **loader** emits every time Routes are added, whilst the size being 0, so either for the first time after construction or after the Router was cleared.

**Source**

packages/shell/src/router/router.ts:173

**shell.Router.**

## [iterator]

*(Readonly Property)*

declared well-known `Symbol.iterator` property. This declaration enforces correct typing when retrieving the `Subscribable` from the well-known `Symbol.observable` method by voiding the inherited `Symbol.iterator`.

**Source**

packages/shell/src/router/router.ts:208

**shell.Router.  
[observable]**

*(Method)*

Well-known `Symbol.observable` method returning a `Subscribable`. The returned `Subscribable` emits the current `State` and every time this changes.

**Example**

Subscribe to upcoming `States`:

```
import { Router } from '@sgrud/shell';
import { from } from 'rxjs';

from(new Router()).subscribe(console.log);
```

**Signature**

```
[observable](): Subscribable<State<string>>
```

**Returns**

A `Subscribable` emitting `States`.

**Source**

packages/shell/src/router/router.ts:286

**shell.Router.****add**

*(Method)*

Overridden **add** method. Invoking this method while supplying a route will **add** the supplied route to the Router after deleting its child Routes from the Router, thereby ensuring that only root routes remain part of the Router.

**Signature**

```
add(route): Router
```

**Returns**

This instance of the Router.

**Parameters**

Name	Type	Description
route	Route<string>	The Route to <b>add</b> to the Router.

**Source**

packages/shell/src/router/router.ts:299

**shell.Router.****baseHref**

*(Readonly Property)*

An absolute **baseHref** for navigation.

**Source**

packages/shell/src/router/router.ts:215

**shell.Router.****connect***(Method)*

**connecting** helper method. Calling this method will **connect** a handler to the global onpopstate event, invoking navigate with the appropriate arguments. This method furthermore allows the properties baseHref, hashBased and outlet to be overridden. Invoking the **connect** method throws an error if called more than once, without invoking the disconnect method in between invocations.

**Throws**

A ReferenceError if already **connected**.

**Signature**

```
connect(this, outlet?, baseHref?, hashBased?): void
```

**Parameters**

Name	Type	Description
this	Mutable<Router>	The Mutable explicit polymorphic this parameter.
outlet	Element   DocumentFragment	The rendering outlet for Routes.
baseHref	string	An absolute baseHref for navigation.
hashBased	boolean	Whether to employ hashBased routing.

**Source**

```
packages/shell/src/router/router.ts:331
```

**shell.Router.****constructor***(Constructor)*

Public Singleton Router class **constructor**. This **constructor** is called once by the Target decorator and sets initial values on this instance. All subsequent calls will return the previously constructed Singleton instance of this class.

**Signature**

```
new Router()
```

**Source**

```
packages/shell/src/router/router.ts:251
```

**shell.Router.****disconnect***(Method)*

**disconnecting** helper method. Calling this method (after calling connect) will **disconnect** the previously connected handler from the global onpopstate event. Further, the arguments passed to connect are revoked, meaning the default values of the properties baseHref, hashBased and outlet are restored. Calling this method without previously connecting the Router throws an error.

**Throws**

A ReferenceError if already **disconnected**.

**Signature**

```
disconnect(this): void
```

**Parameters**

Name	Type	Description
this	Mutable<Router>	The Mutable explicit polymorphic this parameter.

**Source**

```
packages/shell/src/router/router.ts:373
```

**shell.Router.****handle***(Method)*

Implementation of the **handle** method as required by the Queue interface contract. It is called internally by the navigate method after all Queues have been invoked. It is therefore considered the default or fallback Queue and handles the rendering of the supplied state.

**Signature**

```
handle(state, action?): Observable<State<string>>
```

**Returns**

An Observable of the handled State.

**Parameters**

Name	Type	Default value	Description
state	State<string>	undefined	The next State to handle.
action	Action	'push'	The Action to apply to the History.

**Source**

```
packages/shell/src/router/router.ts:396
```

**shell.Router.****hashBased***(Readonly Property)*

Whether to employ **hashBased** routing.

**Source**

```
packages/shell/src/router/router.ts:222
```

**shell.Router.****join***(Method)*

Segment **joining** helper. The supplied segment is converted to a string by spooling to its top-most parent and iterating through all children while concatenating every encountered path. If said path is an (optional) parameter, this portion of the returned string is replaced by the respective Params value.

**Signature**

```
join(segment): string
```

**Returns**

The **joined** Segment in string form.

**Parameters**

Name	Type	Description
segment	Segment<string>	The Segment to be <b>joined</b> .

**Source**

```
packages/shell/src/router/router.ts:438
```

**shell.Router.****lookup***(Method)*

**Lookup** helper method. Calling this method while supplying a selector and optionally an array of routes to iterate will return the **lookupt** Route path for the supplied selector or undefined, if it does not occur within at least one route. When multiple occurrences of the same selector exist, the Route path to its first occurrence is returned.

**Signature**

```
lookup(selector, routes?): undefined | string
```

**Returns**

The **look**uped Route path or undefined.

**Parameters**

Name	Type	Description
selector	string	The Component selector to <b>look</b> up.
routes	Route<string>[]	An array of routes to use for <b>look</b> up.

**Source**

```
packages/shell/src/router/router.ts:472
```

**shell.Router.****match**

(Method)

Main Router **match**ing method. Calling this method while supplying a path and optionally an array of routes will return the first **match**ing Segment or undefined, if nothing **match**es. If no routes are supplied, routes previously added to the Router will be used. The **match** method represents the backbone of this Router class, as it, given a list of routes and a path, will determine whether this path represents a **match** within the list of routes, thereby effectively determining navigational integrity.

**Example**

Test if path 'example/route' **match**es child or route:

```
import { Router } from '@sgrud/shell';

const path = 'example/route';
const router = new Router();

const child = {
  path: 'route'
};

const route = {
  path: 'example',
  children: [child]
};

router.match(path, [child]); // false
router.match(path, [route]); // true
```

**Signature**

```
match(path, routes?): undefined | Segment<string>
```

**Returns**

The first **match**ing Segment or undefined.

**Parameters**

Name	Type	Description
path	string	The path to <b>match</b> routes against.
routes	Route<string>[]	An array of routes to use for <b>match</b> ing.

**Source**

```
packages/shell/src/router/router.ts:526
```

**shell.Router.****navigate**

(Method)

Main **navigate** method. Calling this method while supplying either a path or Segment as navigation target and optional search parameters



will normalize the supplied path by trying to match a respective Segment or directly use the supplied Segment for the next State. This upcoming State is looped through all linked Queues and finally handled by the Router itself to render the resulting, possibly intercepted and mutated State.

#### Throws

An Observable `URIError`, if nothing matches.

#### Signature

```
navigate(target, search?, action?): Observable<State<string>>
```

#### Returns

An Observable of the **navigated** State.

#### Parameters

Name	Type	Default value	Description
target	string   Segment<string>	undefined	Path or Segment to <b>navigate</b> to.
search?	string	undefined	Optional search parameters in string form.
action	Action	'push'	The Action to apply to the History.

#### Source

packages/shell/src/router/router.ts:620

#### shell.Router.

### outlet

(Readonly Property)

The rendering **outlet** for navigated Routes.

#### Source

packages/shell/src/router/router.ts:229

#### shell.Router.

### rebase

(Method)

**rebase** helper method. **rebases** the supplied path against the current baseHref, by either prefixing the baseHref to the supplied path or stripping it, depending on the *prefix* argument.

#### Signature

```
rebase(path, prefix?): string
```

#### Returns

The path **rebased** against the baseHref.

#### Parameters

Name	Type	Default value	Description
path	string	undefined	The path to <b>rebase</b> against the baseHref.
prefix	boolean	true	Whether to prefix or strip the baseHref.

#### Source

packages/shell/src/router/router.ts:679

#### shell.Router.

### spool

(Method)

**spooling** helper method. Given a segment (and whether to rewind), the top-most parent (or deepest child) of the graph-link Segment is returned.

**Signature**

```
spool(segment, rewind?): Segment<string>
```

**Returns**

The **spooled** Segment.

**Parameters**

Name	Type	Default value	Description
segment	Segment<string>	undefined	The Segment to <b>spool</b> .
rewind	boolean	true	Whether to rewind the <b>spool</b> direction.

**Source**

packages/shell/src/router/router.ts:705

**shell.Router.****state**

*(Accessor)*

Getter mirroring the current value of the internal changes BehaviorSubject.

**Signature**

```
get state(): State<string>
```

**Source**

packages/shell/src/router/router.ts:241

**shell.Router.****changes**

*(Private Readonly Property)*

Internally used BehaviorSubject containing and emitting every navigated State.

**Source**

packages/shell/src/router/router.ts:235

**shell.****Router**

*(Namespace)*

Namespace containing types and interfaces used and intended to be used in conjunction with the Singleton Router class.

**See**

Router

**Source**

packages/shell/src/router/router.ts:13, packages/shell/src/router/router.ts:165

**shell.Router.****Action**

*(Type alias)*

Type alias constraining the possible Router **Actions** to 'pop', 'push' and 'replace'. These **Actions** correspond loosely to possible History events.

**Source**

packages/shell/src/router/router.ts:20

**shell.Router.****Left***(Type alias)*

String literal helper type. Represents the **Left**test part of a Route path.

**Example**

**Left** of 'nested/route/path':

```
import { type Router } from '@sgrud/shell';

const left: Router.Left<'nested/route/path'>; // 'nested'
```

**Type parameters**

Name	Type	Description
S	extends string	The Route path string type.

**Source**

packages/shell/src/router/router.ts:36

**shell.Router.****Params***(Type alias)*

Type helper representing the (optional) **Params** of a Route path. By extracting string literals starting with a colon (and optionally ending on a question mark), a union type of a key/value pair for each parameter is created.

**Example**

Extract **Params** from 'item/:id/field/:name?':

```
import { type Router } from '@sgrud/shell';

const params: Router.Params<'item/:id/field/:name?'>;
// { id: string; name?: string; }
```

**Type parameters**

Name	Description
S	The Route path string type.

**Source**

packages/shell/src/router/router.ts:55

**shell.Router.****Queue***(Interface)*

Interface describing the shape of a **Queue**. These **Queues** are run whenever a navigation is triggered and may intercept and mutate the next State or completely block or redirect a navigation.

**See**

Queue

**Implemented by**

Router

**Source**

packages/shell/src/router/router.ts:72

**shell.Router.Queue.****handle***(Method)***handle** method, called when a navigation was triggered.**Signature**

handle(next): Observable&lt;State&lt;string&gt;&gt;

**Returns**An Observable of the **handled** State.**Parameters**

Name	Type	Description
next	State<string>	The next State to be <b>handled</b> .

**Source**

packages/shell/src/router/router.ts:80

**shell.Router.****Segment***(Interface)*

Interface describing the shape of a Router **Segment**. A **Segment** represents a navigated Route and its corresponding Params. As Routes are represented in a tree-like structure and one **Segment** represents one layer within the Route-tree, each **Segment** may have a parent and/or a child. The resulting graph of **Segments** represents the navigated path through the underlying Route-tree.

**Type parameters**

Name	Type	Description
S	extends string = string	The Route path string type.

**Source**

packages/shell/src/router/router.ts:95

**shell.Router.Segment.****child***(Optional Readonly Property)*Optional **child** of this Segment.**Source**

packages/shell/src/router/router.ts:100

**shell.Router.Segment.****params***(Readonly Property)*

Route path Params and their corresponding values.

**Source**

packages/shell/src/router/router.ts:105

**shell.Router.Segment.****parent***(Optional Readonly Property)*Optional **parent** of this Segment.

**Source**

packages/shell/src/router/router.ts:110

**shell.Router.Segment.****route**

*(Readonly Property)*

Route associated with this Segment.

**Source**

packages/shell/src/router/router.ts:115

**shell.Router.****State**

*(Interface)*

Interface describing the shape of a **State** of the Router. **States** correspond to the History, as each navigation results in a new **State** being created. Each navigated **State** is represented by its absolute path its search parameters and a segment as entryptpoint to the graph-like representation of the navigated path through the route-tree.

**Type parameters**

Name	Type	Description
S	extends string = string	The Route path string type.

**Source**

packages/shell/src/router/router.ts:129

**shell.Router.State.****path**

*(Readonly Property)*

Absolute **path** of the State.

**Source**

packages/shell/src/router/router.ts:134

**shell.Router.State.****search**

*(Readonly Property)*

**search** parameters of the State.

**Source**

packages/shell/src/router/router.ts:139

**shell.Router.State.****segment**

*(Readonly Property)*

Segment of the State.

**Source**

packages/shell/src/router/router.ts:144

**shell.**

## RouterLink

*(Class)*

Custom element extending the `HTMLAnchorElement`. This element provides a declarative way to invoke the `navigate` method within the bounds of the `RouterOutlet`, while maintaining compatibility with SSR/SEO aspects of SPAs. This is achieved by rewriting its `href` against the `baseHref` and intercepting the default browser behavior when `onClicked`.

**Example**

A router-link:

```
<a href="/example" is="router-link">Example</a>
```

**See**

Router

**Hierarchy**

- `HTMLAnchorElement`
  - **RouterLink**

**Source**

packages/shell/src/router/link.ts:32

**shell.RouterLink.**

## observedAttributes

*(Static Readonly Property)*

Array of attribute names that should be observed for changes, which will trigger the `attributeChangedCallback`. This element only observes its `href` attribute.

**Source**

packages/shell/src/router/link.ts:39

**shell.RouterLink.**

## attributeChangedCallback

*(Method)*

This method is called whenever this element's `href` attribute is added, removed or changed. The next attribute value is used to determine whether to rebase the `href`.

**Signature**

```
attributeChangedCallback(_name, _prev?, next?): void
```

**Parameters**

Name	Type	Description
<code>_name</code>	<code>string</code>	The <code>_name</code> of the changed attribute (ignored).
<code>_prev?</code>	<code>string</code>	The <code>_previous</code> value of the changed attribute (ignored).
<code>next?</code>	<code>string</code>	The next value of the changed attribute.

**Source**

packages/shell/src/router/link.ts:75

**shell.RouterLink.**

## constructor

*(Constructor)*

Public **constructor** of this custom RouterLink element. This **constructor** is called whenever a new instance this custom element is being rendered into a Document.

**Signature**

```
new RouterLink()
```

**Source**

packages/shell/src/router/link.ts:56

**shell.RouterLink.****onclick**

*(Readonly Property)*

Overridden **onclick** handler, preventing the default browser behavior and invoking navigate instead.

**Signature**

(event): void

**Parameters**

Name	Type	Description
event	MouseEvent	The <b>onclick</b> fired MouseEvent.

**Source**

packages/shell/src/router/link.ts:92

**shell.RouterLink.****router**

*(Private Readonly Property)*

Factored-in **router** property linking the Router.

**Decorator**

Factor

**Source**

packages/shell/src/router/link.ts:49

**shell.****RouterOutlet**

*(Class)*

Custom element extending the HTMLSlotElement. When this element is constructed, it supplies the value of its baseHref attribute and the presence of a hashBased attribute on itself to the Router while connecting the Router to itself. This element should only be used once, as it will be used by the Router as outlet to render the current State.

**Example**

A router-outlet:

```
<slot baseHref="/example" is="router-outlet">Loading...</slot>
```

**See**

Router

**Hierarchy**

- HTMLSlotElement
  - RouterOutlet

**Source**

packages/shell/src/router/outlet.ts:33

**shell.RouterOutlet.****baseHref**

*(Accessor)*

Getter mirroring the **baseHref** attribute of this element.

**Signature**

```
get baseHref(): undefined | string
```

**Source**

```
packages/shell/src/router/outlet.ts:46
```

**shell.RouterOutlet.****constructor**

*(Constructor)*

Public **constructor** of this custom RouterOutlet element. Supplies the value of its baseHref attribute and the presence of a hashBased attribute on itself to the Router while connecting the Router to itself.

**Signature**

```
new RouterOutlet()
```

**Source**

```
packages/shell/src/router/outlet.ts:63
```

**shell.RouterOutlet.****hashBased**

*(Accessor)*

Getter mirroring the presence of a **hashBased** attribute on this element.

**Signature**

```
get hashBased(): boolean
```

**Source**

```
packages/shell/src/router/outlet.ts:53
```

**shell.RouterOutlet.****router**

*(Private Readonly Property)*

Factored-in **router** property linking the Router.

**Decorator**

Factor

**Source**

```
packages/shell/src/router/outlet.ts:41
```

**shell.****component**

*(Const Variable)*

Unique symbol used as property key by the Component decorator to associate the supplied constructor with its wrapper.

**Source**

```
packages/shell/src/component/component.ts:9
```

**shell.****createElement**

*(Function)*

Element factory. Provides JSX runtime compliant bindings creating arrays of bound elementOpen and elementClose calls. This **createElement** factory function is meant to be implicitly imported by the TypeScript transpiler through its JSX bindings and returns an array of bound elementOpen and elementClose function calls, representing the created Element. This array of bound functions can be rendered into an element attached to the Document through the render function.



**See**

render

**Signature**

createElement(type, props?, ref?): Element

**Returns**

An array of bound functions representing the Element.

**Parameters**

Name	Type	Description
type	Function   keyof HTMLElementTagNameMap	The type of Element to create.
props?	Record<string, any>	Any properties to assign to the created Element.
ref?	Key	An optional reference to the created Element.

**Source**

packages/shell/src/component/runtime.ts:116

shell.

**createFragment**

(Function)

JSX fragment factory. Provides a JSX runtime compliant helper creating arrays of bound elementOpen and elementClose calls. This **createFragment** factory function is meant to be implicitly imported by the TypeScript transpiler through its JSX bindings and returns an Element which can be rendered into an element attached to the Document through the render function.

**Signature**

createFragment(props?): Element

**Returns**

An array of bound functions representing the Element.

**Parameters**

Name	Type	Description
props?	Record<string, any>	Any properties to assign to the created Element.

**Source**

packages/shell/src/component/runtime.ts:179

shell.

**customElements**

(Const Variable)

Proxy around the built-in CustomElementRegistry, maintaining a mapping of all registered elements and their corresponding names, which can be queried by calling getName.

**Remarks**

<https://github.com/WICG/webcomponents/issues/566>

**Source**

packages/shell/src/component/registry.ts:13

**shell.**

## html

*(Function)*

Raw **html** rendering helper function. As JSX is pre-processed by the TypeScript transpiler, assigning directly to the `innerHTML` property of an Element will not result in the `innerHTML` to be rendered in the Element. To insert raw **html** into an Element this helper function has to be employed.

**Signature**

```
html(contents, ref?): Element
```

**Returns**

An array of bound functions representing the Element.

**Parameters**

Name	Type	Description
contents	string	The raw <b>html</b> contents to render.
ref?	Key	An optional reference to the created Element.

**Source**

```
packages/shell/src/component/runtime.ts:205
```

**shell.**

## references

*(Function)*

JSX **references** helper. Calling this function while supplying a viable `out let` will return all referenced Elements mapped by their corresponding Keys known to the supplied `out let`. A viable `out let` may be any element which previously was passed as `out let` to the render function.

**Signature**

```
references(out let): Map<Key, Node> | undefined
```

**Returns**

Any **references** known to the supplied `out let`.

**Parameters**

Name	Type	Description
out let	Element   DocumentFragment	The <code>out let</code> to return <b>references</b> for.

**Source**

```
packages/shell/src/component/runtime.ts:226
```

**shell.**

## render

*(Function)*

JSX **rendering** helper. This helper is a small wrapper around the `patch` function and **renders** a Element created through the `createElement` factory into the supplied `out let`.

**See**

```
createElement
```

**Signature**

```
render(out let, element): Node
```

**Returns**

**Rendered** `out let` element.

**Parameters**

Name	Type	Description
out let element	Element   DocumentFragment Element	The out let to <b>render</b> the element into. JSX element to be <b>rendered</b> .

**Source**

packages/shell/src/component/runtime.ts:243

**shell.**

**route**

(Const Variable)

Unique symbol used as property key by the Route decorator to associate the supplied Route configuration with the decorated element.

**Source**

packages/shell/src/router/route.ts:62

## @sgrud/state Module

**@sgrud/state** - The SGRUD State Machine.

The functions and classes found within the **@sgrud/state** module are intended to ease the implementation of Stateful data Stores within applications built upon the SGRUD client libraries. Through wrappers around the IndexedDB and SQLite3 storage Drivers, data will be persisted in every environment. Furthermore, through the employment of Effects, side-effects like retrieving data from external services or dispatching subsequent Actions can be easily achieved.

The **@sgrud/state** module includes a standalone JavaScript bundle which is used to fork a background Thread upon import of this module. This background Thread is henceforth used for State mutation and persistence, independently of the foreground process. Depending on the runtime environment, either a navigator.serviceWorker is registered or a new require('worker\_threads').Worker() NodeJS equivalent will be forked.

**Source**

packages/state/index.ts:1

**state.**

## DispatchEffect

(Class)

Built-in **DispatchEffect** extending the abstract Effect base class. This **DispatchEffect** is automatically implanted when the @sgrud/state module is imported and can therefore be always used in Actions.

**Decorator**

Implant

**See**

Effect

**Hierarchy**

- Effect
  - **DispatchEffect**

**Source**

packages/state/src/effect/dispatch.ts:68

**state.DispatchEffect.**

**constructor**

(Constructor)

Public **constructor** (which should never be called).

**Throws**

A `TypeError` upon construction.

**Signature**

```
new DispatchEffect()
```

**Source**

packages/state/src/effect/effect.ts:71

**state.DispatchEffect.****function**

(Method)

Overridden **function** binding the `DispatchEffect` to the polymorphic `this` of the `StateWorker`.

**Signature**

```
function(this): <T>(handle: Handle, ...action: Action<T>) => Promise<State<T>>
```

**Returns**

This `DispatchEffect` bound to the `StateWorker`.

**Parameters**

Name	Type	Description
<code>this</code>	<code>StateWorker</code>	The explicit polymorphic <code>this</code> parameter.

**Source**

packages/state/src/effect/dispatch.ts:77

**state.****Effect**

(Abstract Class)

Abstract **Effect** base class. When this class is extended and decorated with the `Implant` decorator or implanted through the `StateHandler`, its function will be made available to `Actions` through the global `sgrud.state.effects` namespace.

**Example**

An `importScripts` **Effect**:

```
import { Effect, Implant, type StateWorker, type Store } from '@sgrud/state';

declare global {
  namespace sgrud.state.effects {
    function importScripts(...urls: (string | URL)[]): Promise<void>;
  }
}

@Implant('importScripts')
export class FetchEffect extends Effect {

  public override function(
    this: StateWorker
  ): Store.Effects['importScripts'] {
    return async(...urls) => {
      return importScripts(...urls);
    };
  }
}
```

**Type parameters**

Name	Type	Description
K	extends Effect = Effect	The Effect locate type.

**Hierarchy**

- **Effect**
  - DispatchEffect
  - FetchEffect
  - StateEffect

**Source**

packages/state/src/effect/effect.ts:64

**state.Effect.****constructor**

(Constructor)

Public **constructor** (which should never be called).

**Throws**

A TypeError upon construction.

**Signature**

```
new Effect<K>()
```

**Type parameters**

Name	Type
K	extends Effect = Effect

**Source**

packages/state/src/effect/effect.ts:71

**state.Effect.****function**

(Abstract Method)

Abstract **function** responsible for returning the bound Effect. When an implanted Effect is invoked, it is bound to the polymorphic this of the StateWorker upon invocation. This **function** provides the means of interacting with this bond, as in, utilizing the polymorphic this of the StateWorker to provide the bound Effect, e.g., by utilizing protected properties and methods of the bound-to StateWorker.

**Signature**

```
function(this): typeof effects[K]
```

**Returns**

This Effect bound to the StateWorker.

**Parameters**

Name	Type	Description
this	StateWorker	The explicit polymorphic this parameter.

**Source**

packages/state/src/effect/effect.ts:87

**state.**

## FetchEffect

*(Class)*

Built-in **FetchEffect** extending the abstract **Effect** base class. This **FetchEffect** is automatically implanted when the `@sgrud/state` module is imported and can therefore be always used in Actions.

**Decorator**

Implant

**See**

Effect

**Hierarchy**

- Effect
  - **FetchEffect**

**Source**

packages/state/src/effect/fetch.ts:64

**state.FetchEffect.****constructor***(Constructor)*

Public **constructor** (which should never be called).

**Throws**A `TypeError` upon construction.**Signature**`new FetchEffect()`**Source**

packages/state/src/effect/effect.ts:71

**state.FetchEffect.****function***(Method)*

Overridden **function** binding the **FetchEffect** to the polymorphic `this` of the `StateWorker`.

**Signature**`function(this): (requestInfo: URL | RequestInfo, requestInit?: RequestInit) => Promise<Response>`**Returns**This **FetchEffect** bound to the `StateWorker`.**Parameters**

Name	Type	Description
<code>this</code>	<code>StateWorker</code>	The explicit polymorphic <code>this</code> parameter.

**Source**

packages/state/src/effect/fetch.ts:73

**state.**

## Implant

*(Function)*

The **Implant** decorator, when applied to classes extending the abstract **Effect** base class, implants the decorated class under the `locate` in the global `sgrud.state.effects` namespace to be used within dispatched Actions.

**Example**

An `importScripts` **Effect**:

```
import { Effect, Implant, type StateWorker, type Store } from '@sgrud/state';

declare global {
  namespace sgrud.state.effects {
    function importScripts(...urls: (string | URL)[]): Promise<void>;
  }
}

@Implant('importScripts')
export class ImportScriptsEffect extends Effect {

  public override function(
    this: StateWorker
  ): Store.Effects['importScripts'] {
    return async(...urls) => {
      return importScripts(...urls);
    };
  }
}
```

**See**

StateHandler, Stateful

**Signature**

`Implant<T, K>(locate): (constructor: T) => void`

**Returns**

A class constructor decorator.

**Type parameters**

Name	Type	Description
T	extends () => Effect<K>	An Effect constructor type.
K	extends Effect	The Effect <code>locate</code> type.

**Parameters**

Name	Type	Description
<code>locate</code>	K	The <code>locate</code> to address the Effect by.

**Source**

`packages/state/src/handler/implant.ts:45`

**state.**

**IndexedDB**

*(Class)*

**IndexedDB** Driver. This class provides a facade derived from the built-in Storage interface to IDB Databases within the browser. This class implementing the Driver contract is used as backing storage by the StateWorker, if run in a browser environment.

**See**

Driver

**Implements**

Driver

**Source**

`packages/state/src/driver/indexeddb.ts:11`

**state.IndexedDB.****clear***(Method)*

Removes all key/value pairs, if there are any.

**Signature**

```
clear(): Promise<void>
```

**Returns**

A Promise resolving when this instance was **cleared**.

**Source**

```
packages/state/src/driver/indexeddb.ts:69
```

**state.IndexedDB.****constructor***(Constructor)*

Public IndexedDB **constructor** consuming the name and version used to construct this instance of a Driver.

**Signature**

```
new IndexedDB(name, version)
```

**Parameters**

Name	Type	Description
name	string	The name to address this instance by.
version	string	The version of this instance.

**Source**

```
packages/state/src/driver/indexeddb.ts:38
```

**state.IndexedDB.****getItem***(Method)*

Returns the current value associated with the given key, or null if the given key does not exist.

**Signature**

```
getItem(key): Promise<null | string>
```

**Returns**

A Promise resolving to the current value or null.

**Parameters**

Name	Type	Description
key	string	The key to retrieve the current value for.

**Source**

```
packages/state/src/driver/indexeddb.ts:86
```

**state.IndexedDB.****key***(Method)*

Returns the name of the nth key, or null if n is greater than or equal to the number of key/value pairs.

**Signature**

```
key(index): Promise<null | string>
```



**Returns**

A Promise resolving to the name of the **key** or null.

**Parameters**

Name	Type	Description
index	number	The index of the <b>key</b> to retrieve.

**Source**

packages/state/src/driver/indexeddb.ts:103

**state.IndexedDB.****length**

*(Accessor)*

Returns the number of key/value pairs.

**Signature**

get length(): Promise<number>

**Source**

packages/state/src/driver/indexeddb.ts:21

**state.IndexedDB.****name**

*(Readonly Property)*

The name to address this instance by.

**Source**

packages/state/src/driver/indexeddb.ts:43

**state.IndexedDB.****removeItem**

*(Method)*

Removes the key/value pair with the given key, if a key/value pair with the given key exists.

**Signature**

removeItem(key): Promise<void>

**Returns**

A Promise resolving when the key/value pair was removed.

**Parameters**

Name	Type	Description
key	string	The key to delete the key/value pair by.

**Source**

packages/state/src/driver/indexeddb.ts:122

**state.IndexedDB.****setItem**

*(Method)*

Sets the value of the pair identified by key to value, creating a new key/value pair if none existed for key previously.

**Signature**

```
setItem(key, value): Promise<void>
```

**Returns**

A Promise resolving when the key/value pair was set.

**Parameters**

Name	Type	Description
key	string	The key to set the key/value pair by.
value	string	The value to associate with the key.

**Source**

packages/state/src/driver/indexeddb.ts:140

**state.IndexedDB.****version**

*(Readonly Property)*

The version of this instance.

**Source**

packages/state/src/driver/indexeddb.ts:48

**state.IndexedDB.****database**

*(Private Readonly Property)*

Private **database** used as backing storage to read/write key/value pairs.

**Source**

packages/state/src/driver/indexeddb.ts:16

**state.****SQLite3**

*(Class)*

**SQLite3** Driver. This class provides a facade derived from the built-in Storage interface to **SQLite3** databases under NodeJS. This class implementing the Driver contract is used as backing storage by the StateWorker, if run in a NodeJS environment.

**See**

Driver

**Implements**

Driver

**Source**

packages/state/src/driver/sqlite3.ts:12

**state.SQLite3.****clear**

*(Method)*

Removes all key/value pairs, if there are any.

**Signature**

```
clear(): Promise<void>
```

**Returns**

A Promise resolving when this instance was **cleared**.

**Source**

packages/state/src/driver/sqlite3.ts:76

**state.SQLite3.  
constructor**

(Constructor)

Public SQLite3 **constructor** consuming the name and version used to construct this instance of a Driver.

**Signature**

```
new SQLite3(name, version)
```

**Parameters**

Name	Type	Description
name	string	The name to address this instance by.
version	string	The version of this instance.

**Source**

packages/state/src/driver/sqlite3.ts:39

**state.SQLite3.  
getItem**

(Method)

Returns the current value associated with the given key, or null if the given key does not exist.

**Signature**

```
getItem(key): Promise<null | string>
```

**Returns**

A Promise resolving to the current value or null.

**Parameters**

Name	Type	Description
key	string	The key to retrieve the current value for.

**Source**

packages/state/src/driver/sqlite3.ts:93

**state.SQLite3.  
key**

(Method)

Returns the name of the nth key, or null if n is greater than or equal to the number of key/value pairs.

**Signature**

```
key(index): Promise<null | string>
```

**Returns**

A Promise resolving to the name of the **key** or null.

**Parameters**

Name	Type	Description
index	number	The index of the <b>key</b> to retrieve.

**Source**

packages/state/src/driver/sqlite3.ts:110

**state.SQLite3.****length**

*(Accessor)*

Returns the number of key/value pairs.

**Signature**

get length(): Promise<number>

**Source**

packages/state/src/driver/sqlite3.ts:22

**state.SQLite3.****name**

*(Readonly Property)*

The name to address this instance by.

**Source**

packages/state/src/driver/sqlite3.ts:44

**state.SQLite3.****removeItem**

*(Method)*

Removes the key/value pair with the given key, if a key/value pair with the given key exists.

**Signature**

removeItem(key): Promise<void>

**Returns**

A Promise resolving when the key/value pair was removed.

**Parameters**

Name	Type	Description
key	string	The key to delete the key/value pair by.

**Source**

packages/state/src/driver/sqlite3.ts:127

**state.SQLite3.****setItem**

*(Method)*

Sets the value of the pair identified by key to value, creating a new key/value pair if none existed for key previously.

**Signature**

setItem(key, value): Promise<void>

**Returns**

A Promise resolving when the key/value pair was set.

**Parameters**

Name	Type	Description
key	string	The key to set the key/value pair by.
value	string	The value to associate with the key.

**Source**

packages/state/src/driver/sqlite3.ts:145

**state.SQLite3.****version**

*(Readonly Property)*

The version of this instance.

**Source**

packages/state/src/driver/sqlite3.ts:49

**state.SQLite3.****database**

*(Private Readonly Property)*

Private **database** used as backing storage to read/write key/value pairs.

**Source**

packages/state/src/driver/sqlite3.ts:17

**state.****StateEffect**

*(Class)*

Built-in **StateEffect** extending the abstract Effect base class. This **StateEffect** is automatically implanted when the @sgrud/state module is imported and can therefore be always used in Actions.

**Decorator**

Implant

**See**

Effect

**Hierarchy**

- Effect
  - **StateEffect**

**Source**

packages/state/src/effect/state.ts:61

**state.StateEffect.****constructor**

*(Constructor)*

Public **constructor** (which should never be called).

**Throws**

A TypeError upon construction.

**Signature**

```
new StateEffect()
```

**Source**

packages/state/src/effect/effect.ts:71

**state.StateEffect.****function**

(Method)

Overridden **function** binding the StateEffect to the polymorphic this of the StateWorker.

**Signature**

```
function(this): <T>(handle: Handle) => Promise<State<T> | undefined>
```

**Returns**

This StateEffect bound to the StateWorker.

**Parameters**

Name	Type	Description
this	StateWorker	The explicit polymorphic this parameter.

**Source**

packages/state/src/effect/state.ts:70

**state.****StateHandler**

(Class)

The **StateHandler** Singleton class provides the means to interact with an automatically registered ServiceWorker, when instantiated in a browser environment or, when the **StateHandler** is instantiated within a NodeJS environment, a new `require('worker_threads').Worker()` is forked. Within either of these Threads the StateWorker is executed and handles the deployment of Stores and dispatching Actions against them. The same goes for Effects, whose implantation the StateWorker handles.

The functionality provided by the **StateHandler** is best consumed by applying on of the Stateful or Implant decorators, as those provide easier and higher-level interfaces to the functionality provided by this Singleton class.

**Decorator**

Singleton

**See**

StateWorker

**Source**

packages/state/src/handler/handler.ts:30

**state.StateHandler.****[observable]**

(Static Method)

Static Symbol.observable method returning a Subscribable. The returned Subscribable mirrors the private loader and is used for initializations after the StateHandler has been successfully initialized.

**Example**

Subscribe to the StateHandler:

```
import { StateHandler } from '@sgrud/state';
import { from } from 'rxjs';

from(StateHandler).subscribe(console.log);
```

**Signature**

```
[observable](): Subscribable<StateHandler>
```

**Returns**

A Subscribable emitting this StateHandler.

**Source**

packages/state/src/handler/handler.ts:56

**state.StateHandler.****loader**

*(Static Private Property)*

Private static ReplaySubject used as the StateHandler **loader**. This **loader** emits once after the StateHandler has been successfully initialized.

**Source**

packages/state/src/handler/handler.ts:37

**state.StateHandler.****constructor**

*(Constructor)*

Public StateHandler **constructor**. As the StateHandler is a Singleton class, this **constructor** is only invoked the first time it is targeted by the new operator. Upon this first invocation, the worker property is assigned an instance of the StateWorker Thread while using the supplied source, if any.

**Throws**

A ReferenceError when the environment is incompatible.

**Signature**

```
new StateHandler(source?, scope?)
```

**Parameters**

Name	Type	Description
source?	string	An optional Module source.
scope?	string	An optionally scoped ServiceWorkerRegistration.

**Source**

packages/state/src/handler/handler.ts:95

**state.StateHandler.****deploy**

*(Method)*

Public **deploy** method which defers the **deployment** of the supplied store under the supplied handle to the StateWorker. For convenience, instead of invoking this **deploy** method manually, the Stateful decorator should be considered.

**Signature**

```
deploy<T>(handle, store, state, transient?): Observable<void>
```

**Returns**

An Observable of the Store **deployment**.

**Type parameters**

Name	Type	Description
T	extends Store<any, T>	The extending Store InstanceType.

**Parameters**

Name	Type	Default value	Description
handle	Handle	undefined	The Handle representing the Store.
store	Type<T>	undefined	The Store to <b>deploy</b> under the supplied handle.
state	State<T>	undefined	An initial State for the Store.
transient	boolean	false	Whether the Store is considered transient.

**Source**

packages/state/src/handler/handler.ts:165

**state.StateHandler.****deprecate**

(Method)

Public **deprecate** method which defers to an invocation of the backing **deprecate** method of the StateWorker to **deprecate** the Store represented by the supplied handle.

**Signature**

deprecate(handle): Observable<void>

**Returns**

An Observable of the Store deprecation.

**Parameters**

Name	Type	Description
handle	Handle	The Handle representing the Store.

**Source**

packages/state/src/handler/handler.ts:184

**state.StateHandler.****dispatch**

(Method)

Public **dispatch** method which defers the **dispatching** of the supplied action to the Store represented by the the supplied handle to the StateWorker. For convenience, instead of manually invoking this **dispatch** method manually, the Stateful decorator should be considered.

**Signature**

dispatch<T>(handle, ...action): Observable<State<T>>

**Returns**

An Observable of the resulting State.

**Type parameters**

Name	Type	Description
T	extends Store<any, T>	The extending Store InstanceType.

**Parameters**

Name	Type	Description
handle	Handle	The Handle representing the Store.
...action	Action<T>	A type-guarded Action to <b>dispatch</b> .

**Source**

packages/state/src/handler/handler.ts:202



**state.StateHandler.****implant***(Method)*

Public **implant** method which defers the **implantation** of the supplied effect under the supplied `locate` to the StateWorker. For convenience, instead of invoking this **implant** method manually, the **Implant** decorator should be considered.

**Signature**

```
implant<K>(locate, effect): Observable<void>
```

**Returns**

An Observable of the Store **implantation**.

**Type parameters**

Name	Type	Description
K	extends Effect	The Effect <code>locate</code> type.

**Parameters**

Name	Type	Description
locate	K	The <code>locate</code> to address the Effect by.
effect	() => Effect<K>	The Effect to <b>implant</b> under the <code>locate</code> .

**Source**

packages/state/src/handler/handler.ts:222

**state.StateHandler.****invalidate***(Method)*

Public **invalidate** method which defers to an invocation of the backing **invalidate** method of the StateWorker to **invalidate** the Effect represented by the supplied `locate`.

**Signature**

```
invalidate<K>(locate): Observable<void>
```

**Returns**

An Observable of the Effect invalidation.

**Type parameters**

Name	Type	Description
K	extends Effect	The Effect <code>locate</code> type.

**Parameters**

Name	Type	Description
locate	K	The <code>locate</code> to address the Effect by.

**Source**

packages/state/src/handler/handler.ts:240

**state.StateHandler.****worker***(Readonly Property)*

The **worker** Thread is the main background workhorse, depending on the environment, either a `navigator.serviceWorker` is registered or a new `require('worker_threads').Worker()` NodeJS equivalent will be forked.

**See**

StateWorker

**Source**

packages/state/src/handler/handler.ts:74

**state.StateHandler.****kernel***(Private Readonly Property)*Factored-in **kernel** property linking the Kernel.**Decorator**

Factor

**Source**

packages/state/src/handler/handler.ts:82

**state.****StateWorker***(Class)*

The **StateWorker** is a background Thread which is instantiated by the StateHandler to handle the deployment of Stores and dispatching Actions against them. The same goes for Effects, whose implantation the StateWorker handles.

**Decorator**

Singleton

**See**

StateHandler

**Source**

packages/state/src/worker/index.ts:35

**state.StateWorker.****activate***(Static Private Method)*

Private static **activate** method, called when this StateWorker is instantiated as ServiceWorker in a browser environment upon activation of the ServiceWorker.

**Signature**

activate(event): void

**Parameters**

Name	Type	Description
event	ExtendableEvent	The fired ExtendableEvent.

**Source**

packages/state/src/worker/index.ts:70

**state.StateWorker.****install***(Static Private Method)*

Private static **install** method, called when this StateWorker is instantiated as ServiceWorker in a browser environment upon installation of the ServiceWorker.

**Signature**

```
install(event): void
```

**Parameters**

Name	Type	Description
event	ExtendableEvent	The fired ExtendableEvent.

**Source**

```
packages/state/src/worker/index.ts:81
```

**state.StateWorker.****message**

*(Static Private Method)*

Private static **message** method, called when this StateWorker is instantiated as ServiceWorker in a browser environment upon the reception of messages from the controlling Window.

**Signature**

```
message(event): void
```

**Parameters**

Name	Type	Description
event	ExtendableMessageEvent	The fired ExtendableMessageEvent.

**Source**

```
packages/state/src/worker/index.ts:92
```

**state.StateWorker.****connect**

*(Method)*

Public **connect** method which **connects** this StateWorker to a BusWorker through the supplied socket.

**Remarks**

This method should only be invoked by the StateHandler.

**Signature**

```
connect(socket): Promise<void>
```

**Returns**

A Promise resolving upon socket **connection**.

**Parameters**

Name	Type	Description
socket	MessagePort	A MessagePort to the BusWorker.

**Source**

```
packages/state/src/worker/index.ts:180
```

**state.StateWorker.****constructor**

*(Constructor)*

Public Singleton StateWorker **constructor**. As this is a Singleton **constructor** it is only invoked the first time this StateWorker class is targeted by the new operator. Furthermore this **constructor** returns, depending of the presence of the source parameter, a proxyfied instance of this StateWorker class instead of the actual this reference.

**Remarks**

This method should only be invoked by the StateHandler.

**Signature**

```
new StateWorker(source)
```

**Parameters**

Name	Type	Description
source	null   MessagePort   Client   ServiceWorker	The initial ExtendableMessageEvent source.

**Source**

packages/state/src/worker/index.ts:157

**state.StateWorker.**

**deploy**

(Method)

Public **deploy** method which **deploys** the supplied store under the supplied handle. If the Store is **deployed** transiently, the supplied state is used as initial State. Otherwise, if a previously persisted State exists, it takes precedence over the supplied state. Furthermore, when the supplied Type is already **deployed** and matches the currently **deployed** source code, no action is taken. If the store's sources mismatch, a TypeError is thrown.

**Throws**

A TypeError when the supplied store mismatches.

**Remarks**

This method should only be invoked by the StateHandler.

**Signature**

```
deploy<T>(handle, store, state, transient?): Promise<void>
```

**Returns**

A Promise resolving upon Store **deployment**.

**Type parameters**

Name	Type	Description
T	extends Store<any, T>	The extending Store InstanceType.

**Parameters**

Name	Type	Default value	Description
handle	Handle	undefined	The Handle representing the Store.
store	Type<T>	undefined	The Store to <b>deploy</b> under the supplied handle.
state	State<T>	undefined	An initial State for the Store.
transient	boolean	false	Whether the Store is considered transient.

**Source**

packages/state/src/worker/index.ts:204

**state.StateWorker.**

**deprecate**

(Method)

Public **deprecate** method. When the returned Promise resolves, the deployed Store referenced by the supplied handle is guaranteed to be **deprecated**. Otherwise a ReferenceError is thrown (and therefore the returned Promise rejected).

**Throws**

A `ReferenceError` when no Store could be handled.

**Remarks**

This method should only be invoked by the `StateHandler`.

**Signature**

```
deprecate(handle): Promise<void>
```

**Returns**

A Promise resolving upon Store deprecation.

**Parameters**

Name	Type	Description
handle	Handle	The Handle representing the Store.

**Source**

packages/state/src/worker/index.ts:279

**state.StateWorker.****dispatch**

(Method)

Public **dispatch** method. Invoking this method while supplying a `handle` and a appropriate `action` will apply the supplied Action against the Store deployed under the supplied `handle`. The returned Promise resolves to the resulting new State of the Store after the supplied Action was **dispatched** against it.

**Throws**

A `ReferenceError` when no Store could be handled.

**Remarks**

This method should only be invoked by the `StateHandler`.

**Signature**

```
dispatch<T>(handle, action): Promise<State<T>>
```

**Returns**

A Promise resolving to the resulting State.

**Type parameters**

Name	Type	Description
T	extends <code>Store&lt;any, T&gt;</code>	The extending Store InstanceType.

**Parameters**

Name	Type	Description
handle	Handle	The Handle representing the Store.
action	Action<T>	A type-guarded Action to <b>dispatch</b> .

**Source**

packages/state/src/worker/index.ts:309

**state.StateWorker.****implant**

(Method)

Public **implant** method which **implants** the supplied effect under the supplied `locate` to the global `sgrud.state.effects` namespace. When the supplied Effect is already **implanted** and matches the currently **implanted** source code, no action is taken. If the effect's sources mismatch, a `TypeError` is thrown.

**Throws**

A `TypeError` when the supplied effect mismatches.

**Remarks**

This method should only be invoked by the `StateHandler`.

**Signature**

```
implant<K>(locate, effect): Promise<void>
```

**Returns**

A Promise resolving upon Store **implantation**.

**Type parameters**

Name	Type	Description
K	extends Effect	The Effect locate type.

**Parameters**

Name	Type	Description
locate	K	The locate to address the Effect by.
effect	() => Effect<K>	The Effect to <b>implant</b> under the locate.

**Source**

packages/state/src/worker/index.ts:344

**state.StateWorker.****invalidate**

(Method)

Public **invalidate** method. When the returned Promise resolves, the implanted Effect referenced by the supplied `locate` is guaranteed to be **invalidated**. Otherwise a `ReferenceError` is thrown (and therefore the returned Promise rejected).

**Throws**

A `ReferenceError` when no Effect could be located.

**Remarks**

This method should only be invoked by the `StateHandler`.

**Signature**

```
invalidate<K>(locate): Promise<void>
```

**Returns**

A Promise resolving upon Effect invalidation.

**Type parameters**

Name	Type	Description
K	extends Effect	The Effect locate type.

**Parameters**

Name	Type	Description
locate	K	The locate to address the Effect by.

**Source**

packages/state/src/worker/index.ts:370

**state.StateWorker.**

## **driver**

*(Protected Readonly Property)*

Internal Driver employed as backing data storage. This property contains an instance of either the IndexedDB or the SQLite3 class as abstract facade to either storage provider.

### **Source**

packages/state/src/worker/index.ts:105

**state.StateWorker.**

## **effects**

*(Protected Readonly Property)*

Internal Mapping of Effect locates to their corresponding bound Effects.

### **Source**

packages/state/src/worker/index.ts:111

**state.StateWorker.**

## **proxies**

*(Protected Readonly Property)*

Internal WeakMapping of proxyfied references to this StateWorker to the Effects namespace containing Effects bound to this StateWorker.

### **Source**

packages/state/src/worker/index.ts:118

**state.StateWorker.**

## **remotes**

*(Protected Readonly Property)*

Internal Mapping of Remote BusWorkers to their corresponding proxy of this StateWorker. This Map is used to keep track of the connected Windows and their respective BusWorkers.

### **Source**

packages/state/src/worker/index.ts:126

**state.StateWorker.**

## **states**

*(Protected Readonly Property)*

Internal Mapping of Handles to WeakMapping of States designated by an object reference. This reference either points to the global self reference, if a Store is deployed to be non-transient or, if the opposite applies, to the proxyfied instance of this StateWorker. Through this distinction stores are associated to either a globally shared reference or to a locally contained and transparent Proxy reference to this.

### **Source**

packages/state/src/worker/index.ts:137

**state.StateWorker.**

## **stores**

*(Protected Readonly Property)*

Internal Mapping of deployed Types to their corresponding Handles.

### **Source**

packages/state/src/worker/index.ts:143

**state.StateWorker.**

## proxy

(Private Method)

Private **proxy** method wrapping this StateWorker instance in a Proxy. The resulting Proxy is used to provide distinct this references for each of the connected remotes and intercepts dispatch invocations to provide the globally available sgrud.state.effects namespace.

### Signature

proxy(source): StateWorker

### Returns

A Proxy wrapping the StateWorker.

### Parameters

Name	Type	Description
source	MessagePort   Client   ServiceWorker	The initial ExtendableMessageEvent source.

### Source

packages/state/src/worker/index.ts:386

**state.**

## Stateful

(Function)

The **Stateful** decorator, when applied to classes extending the abstract Store base class, converts those extending classes into type-guarding Store facades implementing only the dispatch and the well-known Symbol.observable methods. This resulting facade provides convenient access to the current and upcoming States of the decorated Store and its dispatch method. The decorated class is deployed under the supplied handle using the supplied state as an initial State. If the Store is to be deployed transiently, the supplied state is guaranteed to be used as initial State. Otherwise, a previously persisted State takes precedence over the supplied state.

### Example

A simple ExampleStore facade:

```
import { Stateful, Store } from '@sgrud/state';

@Stateful('io.github.sgrud.store.example', {
  property: 'default',
  timestamp: Date.now()
})
export class ExampleStore extends Store<ExampleStore> {

  public readonly property!: string;

  public readonly timestamp!: number;

  public async action(property: string): Promise<Store.State<this>> {
    return { ...this, property, timestamp: Date.now() };
  }
}
```

### Example

Subscribe to the ExampleStore facade:

```
import { ExampleStore } from './example-store';

const store = new ExampleStore();
from(store).subscribe(console.log);
// { property: 'default', timestamp: [...] }
```

### Example

Dispatch an Action through the ExampleStore facade:

```
import { ExampleStore } from './example-store';

const store = new ExampleStore();
```



```
store.dispatch('action', ['value']).subscribe(console.log);
// { property: 'value', timestamp: [...] }
```

**See**

StateHandler, Implant

**Signature**

Stateful<T, I>(handle, state, transient?): (constructor: T) => T

**Returns**

A class constructor decorator.

**Type parameters**

Name	Type	Description
T	extends Type<I, T>	A constructor type extending the Type.
I	extends Store<any, I> = InstanceType<T>	The extending Store InstanceType.

**Parameters**

Name	Type	Default value	Description
handle	Handle	undefined	The Handle representing the Store.
state	State<I>	undefined	An initial State for the Store.
transient	boolean	false	Whether the Store is considered transient.

**Source**

packages/state/src/handler/stateful.ts:71

**state.****Store**

(Abstract Class)

Abstract **Store** base class. By extending this **Store** base class and decorating the extending class with the Stateful decorator, the resulting **Store** will become a functional facade implementing only the dispatch and well-known `Symbol.observable` methods. This resulting facade provides convenient access to the current and upcoming States of the **Store** and its dispatch method, while, behind the facade, interactions with the BusHandler to provide an Observable of the State changes and the StateHandler to dispatch any Actions will be handled transparently.

The same functionality can be achieved by manually supplying a **Store** to the StateHandler and subscribing to the changes of that **Store** through the BusHandler while any Actions also have to be passed manually to the StateHandler. But the Stateful decorator should be preferred out of convenience and because invoking the constructor of the **Store** class throws a `TypeError`.

**Example**

A simple ExampleStore facade:

```
import { Stateful, Store } from '@sgrud/state';

@Stateful('io.github.sgrud.store.example', {
  property: 'default',
  timestamp: Date.now()
})
export class ExampleStore extends Store<ExampleStore> {

  public readonly property!: string;

  public readonly timestamp!: number;

  public async action(property: string): Promise<Store.State<this>> {
    return { ...this, property, timestamp: Date.now() };
  }
}
```

**Example**

Subscribe to the ExampleStore facade:

```
import { ExampleStore } from './example-store';

const store = new ExampleStore();
from(store).subscribe(console.log);
// { property: 'default', timestamp: [...] }
```

**Example**

Dispatch an Action through the ExampleStore facade:

```
import { ExampleStore } from './example-store';

const store = new ExampleStore();
store.dispatch('action', ['value']).subscribe(console.log);
// { property: 'value', timestamp: [...] }
```

**Type parameters**

Name	Type	Description
T	extends Store = any	The extending Store InstanceType.

**Source**

packages/state/src/store/store.ts:12, packages/state/src/store/store.ts:164

**state.Store.****[observable]**

(Property)

Well-known Symbol.observable method returning a Subscribable. The returned Subscribable emits all States this Store traverses, i.e., all States that result from dispatching Actions on this Store.

**Throws**

An ReferenceError when not called Stateful.

**Example**

Subscribe to the ExampleStore:

```
import { ExampleStore } from './example-store';

const store = new ExampleStore();
from(store).subscribe(console.log);
```

**Signature**

() : Subscribable<State<T>>

**Returns**

A Subscribable emitting State changes.

**Source**

packages/state/src/store/store.ts:184

**state.Store.****constructor**

(Constructor)

**Throws**

A TypeError upon construction.

**Signature**

new Store<T>()

**Type parameters**

Name	Type
T	extends Store<any, T> = any

**Source**

packages/state/src/store/store.ts:189

**state.Store.****dispatch**

(Method)

The **dispatch** method provides a facade to **dispatch** an Action through the StateHandler when this Store was decorated with the Stateful decorator, otherwise calling this method will throw an ReferenceError.

**Throws**

An ReferenceError when not called Stateful.

**Example**

Dispatch an Action to the ExampleStore:

```
import { ExampleStore } from './example-store';

const store = new ExampleStore();
store.dispatch('action', ['value']).subscribe();
```

**Signature**

dispatch(...action): Observable<State<T>>

**Returns**

An Observable of the resulting State.

**Parameters**

Name	Type	Description
...action	Action<T>	A type-guarded Action to <b>dispatch</b> .

**Source**

packages/state/src/store/store.ts:212

**state.****Store**

(Namespace)

The **Store** namespace contains types and interfaces used and intended to be used in conjunction with the abstract Store class.

**See**

Store

**Source**

packages/state/src/store/store.ts:12, packages/state/src/store/store.ts:164

**state.Store.****Action**

(Type alias)

This Store **Action** helper type represents the signatures of all available **Actions** of any given Store by extracting all methods from the given Store that return a promisified State of that given Store. This State is interpreted as the next State after this **Action** was invoked.

**Type parameters**

Name	Type	Description
T	extends Store	The extending Store InstanceType.

**Source**

packages/state/src/store/store.ts:24

**state.Store.****Driver**

*(Type alias)*

The **Driver** helper type is a promisified variant of the built-in Storage type. This type is utilized by the StateWorker where it represents one of the available Storage **Drivers**.

**Implemented by**

IndexedDB, SQLite3

**Source**

packages/state/src/store/store.ts:40

**state.Store.****Effect**

*(Type alias)*

The **Effect** helper type represents a keyof the Effects map.

**Source**

packages/state/src/store/store.ts:50

**state.Store.****Effects**

*(Type alias)*

The **Effects** helper type represents the typeof the globally available sgrud.state.effects namespace.

**Source**

packages/state/src/store/store.ts:56

**state.Store.****State**

*(Type alias)*

The Store **State** helper type represents the current **State** of any given Store by extracting all properties (and dropping any methods) from that given Store.

**Type parameters**

Name	Type	Description
T	extends Store	The extending Store InstanceType.

**Source**

packages/state/src/store/store.ts:65

**state.Store.****States**

*(Type alias)*

The **States** helper type represents the traversal of Stores.

**Source**

packages/state/src/store/store.ts:77

**state.Store.****Type**

*(Interface)*

Interface describing the **Type**, i.e., static constructable context, of classes extending the abstract Store base class.

**Type parameters**

Name	Type	Description
T	extends Store	The extending Store InstanceType.

**Hierarchy**

- Required<typeof Store>
  - **Type**

**Source**

packages/state/src/store/store.ts:85

**state.Store.Type.****constructor**

*(Constructor)*

Overridden and concretized constructor signature.

**Signature**

new Type()

**Source**

packages/state/src/store/store.ts:85

**state.Store.Type.****prototype**

*(Readonly Property)*

Overridden prototype signature.

**Source**

packages/state/src/store/store.ts:90

## Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	99.97	96.11	99.63	99.97	
bus/src/bus	99.77	97.61	100	99.77	
bus.ts	99.61	92.85	100	99.61	1
querier.ts	100	100	100	100	
transfer.ts	100	100	100	100	
bus/src/handler	100	97.56	100	100	
handler.ts	100	95.65	100	100	133
observe.ts	100	100	100	100	
publish.ts	100	100	100	100	
stream.ts	100	100	100	100	
core/src/http	100	100	100	100	
http.ts	100	100	100	100	
proxy.ts	100	100	100	100	
transit.ts	100	100	100	100	
core/src/kernel	100	98.05	100	100	
kernel.ts	100	97.14	100	100	345,376
semver.ts	100	100	100	100	
core/src/linker	100	94.44	100	100	
factor.ts	100	100	100	100	
linker.ts	100	90	100	100	35
target.ts	100	100	100	100	
core/src/super	100	94.73	100	100	
provide.ts	100	100	100	100	
provider.ts	100	100	100	100	
registry.ts	100	93.75	100	100	95
core/src/thread	100	95.83	100	100	
spawn.ts	100	93.33	100	100	75
thread.ts	100	100	100	100	
transfer.ts	100	100	100	100	
core/src/utility	100	92.15	95.83	100	
assign.ts	100	100	100	100	
pluralize.ts	100	100	100	100	
singleton.ts	100	78.94	80	100	27,49
symbols.ts	100	100	100	100	
type-of.ts	100	100	100	100	
data/src/model	100	99.15	100	100	
enum.ts	100	83.33	100	100	10
model.ts	100	100	100	100	
data/src/querier	100	100	100	100	
http.ts	100	100	100	100	
querier.ts	100	100	100	100	
data/src/relation	100	100	100	100	
has-many.ts	100	100	100	100	
has-one.ts	100	100	100	100	
property.ts	100	100	100	100	
shell/src/component	99.88	97.45	100	99.88	
attribute.ts	98.73	87.5	100	98.73	64
component.ts	100	96.29	100	100	177
fluctuate.ts	100	100	100	100	
reference.ts	100	100	100	100	
registry.ts	100	100	100	100	
runtime.ts	100	100	100	100	
shell/src/jsx-runtime	100	100	100	100	
index.ts	100	100	100	100	
shell/src/queue	100	94.04	100	100	
catch.ts	100	92.72	100	100	86,195,201,228
queue.ts	100	100	100	100	
resolve.ts	100	96.42	100	100	96
shell/src/router	100	95.31	100	100	
link.ts	100	85.71	100	100	57
outlet.ts	100	85.71	100	100	64
route.ts	100	100	100	100	
router.ts	100	95.95	100	100	252,398,483,664
state/src/driver	100	98.55	100	100	
indexeddb.ts	100	97.61	100	100	11
sqlite3.ts	100	100	100	100	
state/src/effect	100	78.12	100	100	
dispatch.ts	100	83.33	100	100	68
effect.ts	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
fetch.ts	100	83.33	100	100	64
state.ts	100	63.63	100	100	61,73-76
transfer.ts	100	87.5	100	100	14
state/src/handler	100	92.45	100	100	
handler.ts	100	90	100	100	110,115,122,149
implant.ts	100	100	100	100	
stateful.ts	100	100	100	100	
state/src/store	100	92.3	100	100	
store.ts	100	100	100	100	
transfer.ts	100	90	100	100	13

# Index

**[hasMany]**

data.Model, 89

**[hasOne]**

data.Model, 89

**[iterator]**

shell.Router, 132

**[observable]**

bus.Bus, 22

bus.BusHandler, 26

core.Kernel, 45

core.Transit, 66

data.Model, 89

shell.Router, 132, 133

state.StateHandler, 158

state.Store, 170

**[property]**

data.Model, 90

**[provide]**

bus.BusQuerier, 30

core.Provider, 57

core.Proxy, 58

core.Transit, 66

data.HttpQuerier, 79

data.Querier, 112

shell.CatchQueue, 118

shell.Queue, 126

shell.ResolveQueue, 129

**[toStringTag]**

data.Model, 96

**Action**

shell.Router, 138

state.Store, 171

**activate**

state.StateWorker, 162

**add**

shell.Router, 133

**adoptedCallback**

shell.Component, 120

**Alias**

core, 37

**array**

core.TypeOf, 67

**Assign**

core, 38

**assign**

core, 74

data.Model, 90

**Attribute**

shell, 116

**attributeChangedCallback**

shell.Component, 121

shell.RouterLink, 142

**baseHref**

shell.Router, 133

shell.RouterOutlet, 143

**bin**

global, 16

**boolean**

core.TypeOf, 68

**Bus**

bus, 21, 25

**bus**

global, 21

**BusHandler**

bus, 26

**BusQuerier**

bus, 29

**BusWorker**

bus, 31

**cached**

core.Registry, 61

**caches**

core.Registry, 61

**Catch**

shell, 116, 117

**CatchQueue**

shell, 117

**changes**

bus.BusWorker, 33

core.Kernel, 49

core.Transit, 67



- data.Model, 96
- shell.Router, 138
- child**
  - shell.Router.Segment, 140
- children**
  - shell.Route, 131
- clear**
  - data.Model, 90
  - state.IndexedDB, 152
  - state.SQLite3, 154
- commit**
  - bus.BusQuerier, 30
  - data.HttpQuerier, 79
  - data.Model.Type, 102
  - data.Model, 81, 91
  - data.Querier, 112
- complete**
  - bus.Bus, 22
- Component**
  - shell, 119, 120
- component**
  - shell, 144
  - shell.Route, 131
- Conjunction**
  - data.Model.Filter, 98
- conjunction**
  - data.Model.Filter.Expression, 98
- connect**
  - shell.Router, 134
  - state.StateWorker, 163
- connectedCallback**
  - shell.Component, 121
- construct**
  - bin, 16
- constructor**
  - bus.Bus, 22
  - bus.BusHandler, 27
  - bus.BusQuerier, 30
  - bus.BusWorker, 32
  - core.Http, 44
  - core.Kernel, 46
  - core.Linker, 53
  - core.Provider, 57
  - core.Registry, 60
  - core.Transit, 66
  - core.TypeOf, 74
  - data.Enum, 76
  - data.HttpQuerier, 79
  - data.Model.Type, 103
  - data.Model, 92
  - shell.CatchQueue, 118
  - shell.ResolveQueue, 129
  - shell.Router, 134
  - shell.RouterLink, 142
  - shell.RouterOutlet, 144
  - state.DispatchEffect, 147
  - state.Effect, 149
  - state.FetchEffect, 150
  - state.IndexedDB, 152
  - state.SQLite3, 155
  - state.StateEffect, 157
  - state.StateHandler, 159
  - state.StateWorker, 163
  - state.Store.Type, 173
  - state.Store, 170
- core**
  - global, 37
- created**
  - data.Model, 92
- createElement**
  - shell, 144
- createFragment**
  - shell, 145
- customElements**
  - shell, 145
- CustomElementTagName**
  - shell, 123
- data**
  - global, 76
- database**
  - state.IndexedDB, 154
  - state.SQLite3, 157
- date**
  - core.TypeOf, 68
- delete**
  - core.Http, 39
  - data.Model, 92
- deleteAll**
  - data.Model.Type, 103
  - data.Model, 82
- deleteOne**
  - data.Model.Type, 104
  - data.Model, 82
- deploy**
  - state.StateHandler, 159
  - state.StateWorker, 164
- deprecate**

- state.StateHandler, 160
- state.StateWorker, 164
- Digest**
  - core.Kernel, 50
- digest**
  - core.Kernel.Module, 51
- dir**
  - data.Model.Filter.Params, 100
- disconnect**
  - shell.Router, 134
- disconnectedCallback**
  - shell.Component, 121
- dispatch**
  - state.StateHandler, 160
  - state.StateWorker, 165
  - state.Store, 171
- DispatchEffect**
  - state, 147
- Driver**
  - state.Store, 172
- driver**
  - state.StateWorker, 167
- Effect**
  - state, 148
  - state.Store, 172
- Effects**
  - state.Store, 172
- effects**
  - state.StateWorker, 167
- Element**
  - shell.JSX, 124
- endpoint**
  - data.HttpQuerier, 80
- entity**
  - data.Model.Filter.Expression, 99
  - data.Model, 96
- Enum**
  - data, 76
- enumerate**
  - data, 114
- error**
  - bus.Bus, 23
- exports**
  - core.Kernel.Module, 51
  - core.Kernel.WebDependency, 52
- Expression**
  - data.Model.Filter, 98
- expression**
- data.Model.Filter.Params, 100
- Factor**
  - core, 38
- FetchEffect**
  - state, 150
- Field**
  - data.Model, 97
- Filter**
  - data.Model, 98
- find**
  - data.Model, 93
- findAll**
  - data.Model.Type, 105
  - data.Model, 83
- findOne**
  - data.Model.Type, 106
  - data.Model, 84
- Fluctuate**
  - shell, 123
- fluctuationChangedCallback**
  - shell.Component, 121
- function**
  - core.TypeOf, 69
  - state.DispatchEffect, 148
  - state.Effect, 149
  - state.FetchEffect, 150
  - state.StateEffect, 158
- get**
  - core.Http, 40
  - core.Linker, 54
  - core.Registry, 60
- getAll**
  - core.Linker, 54
- getItem**
  - state.IndexedDB, 152
  - state.SQLite3, 155
- Graph**
  - data.Model, 101
- Handle**
  - bus.Bus, 25
- handle**
  - bus.Bus, 23
  - bus.BusQuerier, 31
  - core.Http.Handler, 44
  - core.Http, 43
  - core.Proxy, 58
  - core.Transit, 66

- shell.CatchQueue, 118
- shell.Queue, 126
- shell.ResolveQueue, 129
- shell.Router, 135
- shell.Router.Queue, 140
- handleErrors**
  - shell.CatchQueue, 119
- Handler**
  - core.Http, 44
- hashBased**
  - shell.Router, 135
  - shell.RouterOutlet, 144
- HasMany**
  - data, 77
- hasMany**
  - data, 115
- HasOne**
  - data, 77
- hasOne**
  - data, 115
- head**
  - core.Http, 40
- html**
  - shell, 146
- HTMLElementTagName**
  - shell, 124
- Http**
  - core, 39, 44
- HttpQuerier**
  - data, 78
- Implant**
  - state, 150
- implant**
  - state.StateHandler, 161
  - state.StateWorker, 165
- imports**
  - core.Kernel, 49
- IndexedDB**
  - state, 151
- inmod**
  - core.Kernel, 46
- install**
  - state.StateWorker, 162
- IntrinsicElements**
  - shell.JSX, 125
- invalidate**
  - state.StateHandler, 161
  - state.StateWorker, 166
- join**
  - shell.Router, 135
- JSX**
  - shell, 124
- Kernel**
  - core, 45, 50
- kernel**
  - state.StateHandler, 162
- Key**
  - shell.JSX, 125
- key**
  - state.IndexedDB, 152
  - state.SQLite3, 155
- kickstart**
  - bin, 17
- Left**
  - shell.Router, 139
- length**
  - state.IndexedDB, 153
  - state.SQLite3, 156
- Linker**
  - core, 52
- loader**
  - bus.BusHandler, 27
  - shell.Router, 132
  - state.StateHandler, 159
- loaders**
  - core.Kernel, 49
- lookup**
  - shell.Router, 135
- match**
  - shell.Router, 136
- Merge**
  - core, 55
- message**
  - state.StateWorker, 163
- Model**
  - data, 81, 97
- modified**
  - data.Model, 93
- Module**
  - core.Kernel, 50
- Mutable**
  - core, 55
- name**
  - core.Kernel.Module, 51

- state.IndexedDB, 153
- state.SQLite3, 156
- navigate**
  - shell.Router, 136
- next**
  - bus.Bus, 23
- nodeModules**
  - core.Kernel, 47
- null**
  - core.TypeOf, 69
- number**
  - core.TypeOf, 70
- object**
  - core.TypeOf, 70
- Observe**
  - bus, 34
- observe**
  - bus.Bus, 24
  - bus.BusHandler, 27
  - bus.BusWorker, 32
- observedAttributes**
  - shell.Component, 122
  - shell.RouterLink, 142
- observedFluctuations**
  - shell.Component, 122
- observedReferences**
  - shell.Component, 122
- onclick**
  - shell.RouterLink, 143
- Operation**
  - data.Querier, 114
- Operator**
  - data.Model.Filter, 99
- outlet**
  - shell.Router, 137
- page**
  - data.Model.Filter.Params, 100
- Params**
  - data.Model.Filter, 99
  - shell.Router, 139
- params**
  - shell.Router.Segment, 140
- parent**
  - shell.Router.Segment, 140
- patch**
  - core.Http, 41
- Path**
  - data.Model, 102
- path**
  - shell.Route, 131
  - shell.Router.State, 141
- plural**
  - data.Model, 96
- pluralize**
  - core, 75
- post**
  - core.Http, 41
- postbuild**
  - bin, 18
- prioritize**
  - bus.BusQuerier, 31
  - data.HttpQuerier, 80
- priority**
  - bus.BusQuerier, 31
  - data.HttpQuerier, 80
  - data.Querier, 113
- process**
  - core.TypeOf, 70
- promise**
  - core.TypeOf, 71
- Property**
  - data, 110, 111
- property**
  - data, 115
- prototype**
  - data.Model.Type, 106
  - state.Store.Type, 173
- Provide**
  - core, 55
- provide**
  - core, 75
- Provider**
  - core, 56, 57
- proxies**
  - state.StateWorker, 167
- Proxy**
  - core, 58
- proxy**
  - state.StateWorker, 168
- Publish**
  - bus, 35
- publish**
  - bus.Bus, 24
  - bus.BusHandler, 28
  - bus.BusWorker, 32
- put**
  - core.Http, 42

**Querier**

data, 111, 113

**Queue**

shell, 125

shell.Router, 139

**rebase**

shell.Router, 137

**Reference**

shell, 126

**referenceChangedCallback**

shell.Component, 122

**references**

shell, 146

**regex**

core.TypeOf, 71

**Registration**

core, 59

**Registry**

core, 59

**remotes**

state.StateWorker, 167

**removeItem**

state.IndexedDB, 153

state.SQLite3, 156

**render**

shell, 146

**renderComponent**

shell.Component, 123

**Request**

core.Http, 45

**request**

core.Http, 43

**requests**

core.Transit, 67

**require**

core.Kernel, 47

**required**

shell.ResolveQueue, 129

**Resolve**

shell, 127

**resolve**

core.Kernel, 48

**resolved**

shell.ResolveQueue, 129

**ResolveQueue**

shell, 128

**Response**

core.Http, 45

**result**

data.Model.Filter.Results, 101

**Results**

data.Model.Filter, 101

**Route**

shell, 130

**route**

shell, 147

shell.Router.Segment, 141

**Router**

shell, 132, 138

**router**

shell.CatchQueue, 119

shell.RouterLink, 143

shell.RouterOutlet, 144

**RouterLink**

shell, 142

**RouterOutlet**

shell, 143

**runtimify**

bin, 19

**save**

data.Model, 94

**saveAll**

data.Model.Type, 106

data.Model, 85

**saveOne**

data.Model.Type, 107

data.Model, 85

**script**

core.Kernel, 48

**search**

data.Model.Filter.Params, 100

shell.Router.State, 141

**Segment**

shell.Router, 140

**segment**

shell.Router.State, 141

**semver**

core, 75

**serialize**

data.Model.Type, 108

data.Model, 86, 94

**set**

core.Registry, 61

**setItem**

state.IndexedDB, 153

state.SQLite3, 156

**sgrudDependencies**

core.Kernel.Module, 51

**Shape**

data.Model, 102

**shell**

global, 115

**shimmed**

core.Kernel, 50

**Singleton**

core, 62

**size**

data.Model.Filter.Params, 100

**slots**

shell.Route, 131

**sort**

data.Model.Filter.Params, 101

**Spawn**

core, 62

**spool**

shell.Router, 137

**SQLite3**

state, 154

**State**

shell.Router, 141

state.Store, 172

**state**

shell.Router, 138

global, 147

**StateEffect**

state, 157

**Stateful**

state, 168

**StateHandler**

state, 158

**States**

state.Store, 172

**states**

state.StateWorker, 167

**StateWorker**

state, 162

**static**

data.Model, 97

**Store**

state, 169, 171

**stores**

state.StateWorker, 167

**Stream**

bus, 36

**streams**

bus.BusWorker, 34

**string**

core.TypeOf, 72

**styles**

shell.Component, 123

**subscribe**

bus.Bus, 24

**Symbol**

core, 63

**Target**

core, 63, 64

**template**

shell.Component, 123

**test**

core.TypeOf, 73

**Thread**

core, 64, 65

**total**

data.Model.Filter.Results, 101

**Transit**

core, 65

**trapped**

shell.CatchQueue, 118

**traps**

shell.CatchQueue, 119

**treemap**

data.Model.Type, 109

data.Model, 87, 95

**Type**

data.Model, 102

data.Querier, 114

state.Store, 173

**type**

data.Model, 97

**TypeOf**

core, 67

**types**

bus.BusQuerier, 31

data.HttpQuerier, 80

data.Querier, 113

**undefined**

core.TypeOf, 72

**universal**

bin, 20

**unpkg**

core.Kernel.Module, 51

core.Kernel.WebDependency, 52

**unravel**

data.Model.Type, 109

data.Model, 88

#### **uplink**

bus.BusHandler, 28

bus.BusWorker, 33

#### **uplinks**

bus.BusWorker, 34

#### **url**

core.TypeOf, 73

#### **uuid**

data.Model, 95

#### **valuate**

data.Model.Type, 110

data.Model, 88

#### **Value**

bus.Bus, 25

#### **Variables**

data.Querier, 114

#### **verify**

core.Kernel, 49

#### **version**

core.Kernel.Module, 51

state.IndexedDB, 154

state.SQLite3, 157

#### **webDependencies**

core.Kernel.Module, 52

#### **WebDependency**

core.Kernel, 52

#### **window**

core.TypeOf, 73

#### **worker**

bus.BusHandler, 29

state.StateHandler, 161